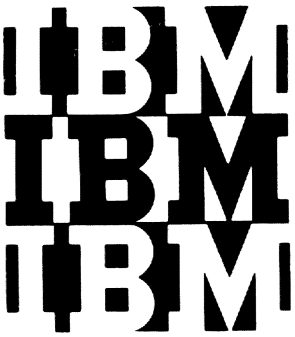


File No. S360-37 (TSS)
GY28-2014-2

Program Logic

Version 8.1

IBM System/360 Time Sharing System

Program Control System

This publication describes the internal logic of the Program Control System of IBM System/360 Time Sharing System. It offers facilities within the Command System that provide the user with program control and checkout aids at both load and execution time.

Program logic manuals are intended for use by IBM customer engineers involved in program maintenance and by system programmers involved in altering the program design. Program logic information is not necessary for program operation and use; therefore, distribution of this manual is limited to people with program maintenance or modification responsibilities.

PREFACE

This publication provides customer engineers and other technical personnel with information describing the internal organization and logic of the Program Control System (PCS). This material is divided into four sections and eight appendixes.

Section 1 is an introduction to PCS and describes it as an operating entity, including its relationship to the Time Sharing System and especially the Command Language Interpreter.

Section 2 concentrates on the interactions of the various PCS routines. It defines their interrelationships as well as their use of communication areas so as to provide a frame of reference for the details that follow.

Section 3 provides a detailed description of the functions and internal logic of the individual PCS routines.

Section 4 consists of flowcharts that are compatible with the level of detail supplied in Section 3. Thus, they are arranged in the same order (i.e., by routine identification - CZA..).

The appendixes contain supplementary material which can be referred to while using other portions of the manual. Appendixes A and B provide lists of PCS routine identifications and mnemonics. Appendix C is a table detailing PCS routines and their calling conditions. Appendix F describes, in detail, the formats and contents of the tables used by PCS routines. Appendix G gives examples of PCS output formats and Appendix H details the PCS processing limitations.

Prerequisite Reading

Effective use of this manual is based on an understanding of TSS/360 as discussed in:

IBM System/360 Time Sharing System: Concepts and Facilities, Form C28-2003

IBM System/360 Time Sharing System: System Logic Summary, Form Y28-2009

IBM System/360 Time Sharing System: Command System User's Guide, Form C28-2001

Third Edition (September 1971)

This is a major revision of, and obsoletes, GY28-2014-1, and Technical Newsletters GY28-3009, GY28-3114 GN28-3115 and GN28-3162. This edition is current with Version 8, Modification 1, and remains in effect for all subsequent versions or modifications of IBM System/360 Time Sharing System unless otherwise indicated. Significant changes or additions to this publication will be provided in new editions or Technical Newsletters. Before using this publication in connection with the operation of IBM systems, refer to IBM System/360 Time Sharing System: Addendum, GC28-2043-12, for the editions of publications that are applicable and current.

This edition of the Program Control System PLM incorporates the TNLs mentioned above and reflects the following additional changes to the Program Control System: Offset from location 0 is assumed to be an offset from the qualified name if a qualification is in effect. Besides offset and length, the type of a display or dump may now be specified; that is hexadecimal, binary, floating-point, character, symbolic, or integer. The Program Module Dictionary (PMD) has been changed.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form is provided at the back of this publication for readers' comments. If the form has been removed, comments may be addressed to IBM Corporation, Time Sharing Systems Publications, Department 561, P.O. Box 344, 2651 Strang Boulevard, Yorktown Heights, N.Y. 10598

CONTENTS

ILLUSTRATIONS

FIGURES

TABLES

CHARTS

PURPOSE OF THE PROGRAM CONTROL SYSTEM

The Program Control System (PCS) is an integral component of the TSS/360 Command System, offering facilities within the command system that provide the user with program control and checkout aids at both load and execution time. PCS commands are used to:

- Request at any time during execution of a program, display of the contents of data fields, instruction locations, and registers.
- Modify the contents of the user's virtual storage.
- Specify locations within his program where execution is to be stopped or started. When execution stops, the user can issue additional commands before he resumes execution.
- Establish logical (i.e., true or false) conditions that allow or inhibit the execution of other commands.

These control and checkout services are requested by means of the PCS commands: AT, BRANCH, CALL, DISPLAY, DUMP, GO, IF, QUALIFY, REMOVE, SET, and STOP. These commands and their usage are defined in the Command System User's Guide.

RELATIONSHIP OF PCS TO TSS/360 PROGRAMMING SYSTEM

PCS is an extension of the command analyzer and executor (CA&E) routines within the command system. PCS is present in initial virtual storage (IVM) at logon time and is called by CA&E when CA&E recognizes a user-issued command requesting PCS services. PCS does not directly service interrupts nor does it execute privileged instructions.

PCS OPERATIONAL CHARACTERISTICS

PCS is loaded as part of initial virtual storage by TSS STARTUP. Its routines are reenterable and are dynamically paged into and out of main storage as needed. The services of PCS are always available to the user.

PCS commands and statements may be issued in conversational or nonconversational mode. All commands are obtained from SYSIN, without distinction as to mode. In the conversational mode, a syntax check is made and symbolic references are validated. If syntactic errors or references to undefined symbols are detected, appropriate messages are written on SYSOUT. In the non-conversational mode, the same checks as in the conversational mode are made but diagnostics will cause the statement to be ignored. Diagnostics are delivered to the task's SYSOUT data set together with the PCS output, and incorrect commands are ignored. Output from the DUMP command is always written to the PCSOUT data set for subsequent printing.

PCS statements may require immediate or deferred execution depending on the presence of an AT command. Those statements which do not contain an AT command are executed immediately. Those statements which do contain an AT command are processed during object-program execution upon arrival at the location specified in the AT command. This latter deferred type of command execution is said to be dynamic and is accomplished by the insertion of a PCSVC supervisor call in the issuing task's program at the points specified in the AT command.

PCS INTERFACE WITH OTHER TSS/360 SERVICE PROGRAMS

PCS interfaces with other system modules in performing its task (Figure 1). All linkages between PCS and such system modules are type-I linkages.

COMMAND ANALYZER AND EXECUTOR (CA&E)

The command analyzer and executor (CA&E) serves as the primary link between PCS and the user's program.

CA&E obtains user's PCS statements from SYSIN in exactly the same way as for other commands. CA&E scans the statement and, detecting a PCS command, calls the appropriate PCS module to perform the required processing.

CA&E also serves as a control program for PCS, exercising control over the flow of work between PCS phases. Upon completion of the processing of a phase, PCS returns control to CA&E, unless otherwise noted (Figure 3).

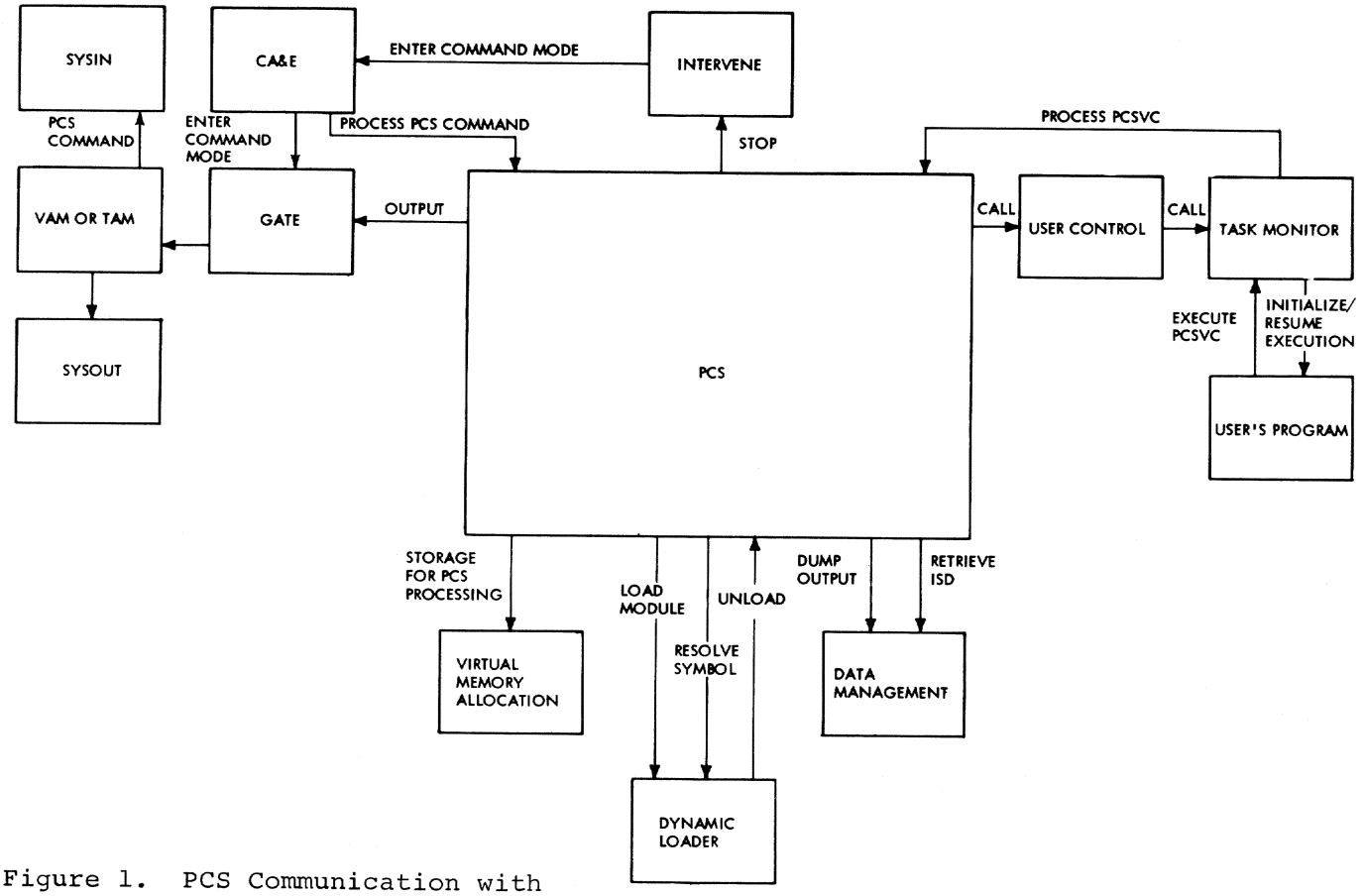


Figure 1. PCS Communication with TSS/360 Service Routines

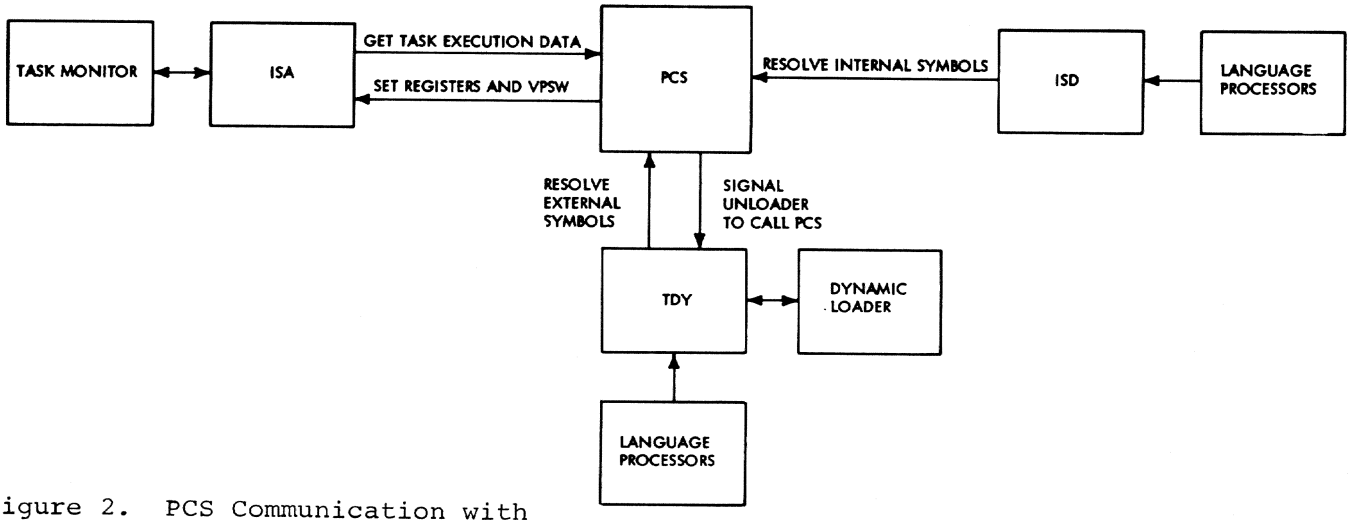


Figure 2. PCS Communication with TSS/360 Service Routines via Common Areas

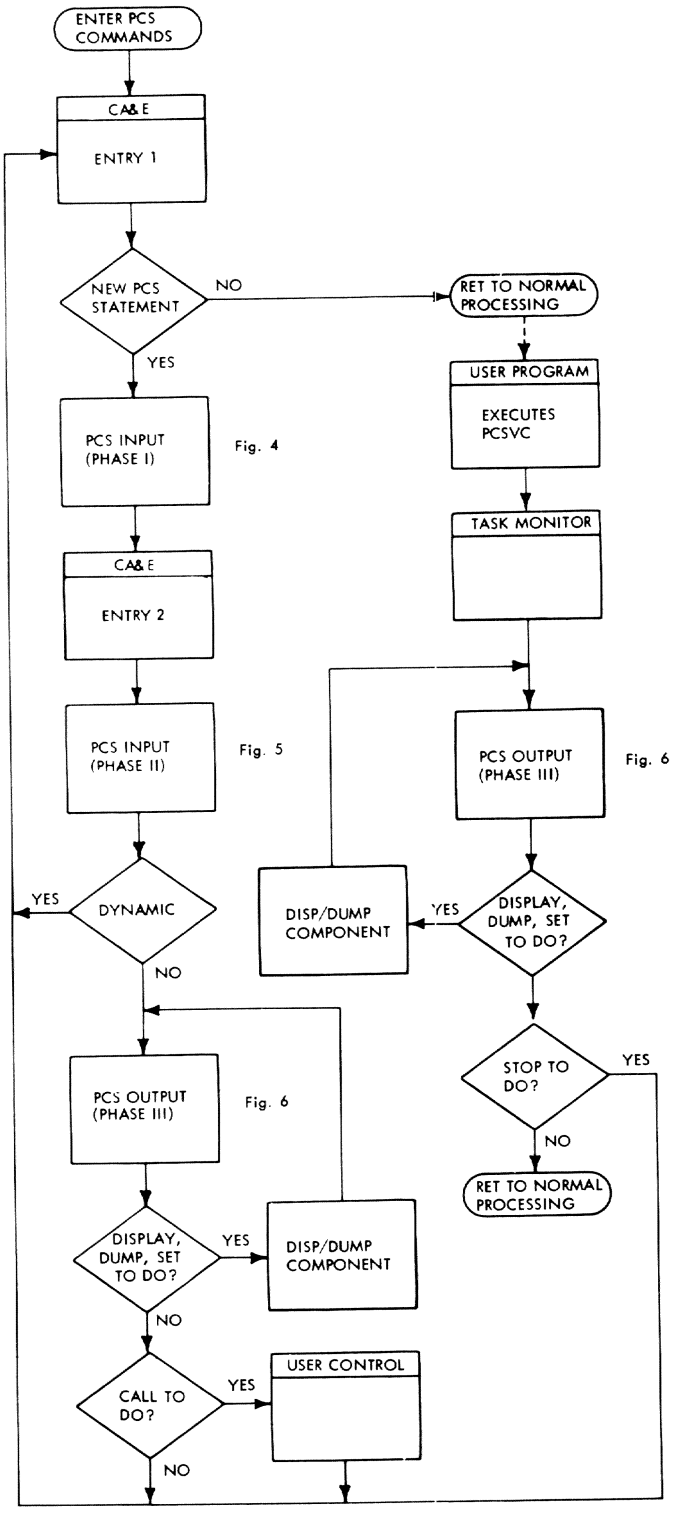


Figure 3. Overview of PCS Components

USER CONTROL ROUTINE

PCS, when processing the CALL command, gives control to the User Control service routine to initiate execution of the user's program.

TASK MONITOR

The task monitor is called to initiate program execution when PCS processes a CALL command. PCS calls the task monitor indirectly, via the User Control service routine. Upon encountering a PCSVC in the user program, the task monitor calls PCS to provide the processing required to complete the actions in a dynamic statement.

INTERVENE

During the processing of a dynamic statement, PCS determines if the task should be placed back in the command mode. This could result from the processing of a dynamic STOP command or from an error condition recognized by PCS. A task is placed in command mode when control is given to INTERVENE to halt the execution of the user program and give control to CA&E.

DATA MANAGEMENT

Data management facilities are used for the retrieval of internal symbol dictionaries (ISD). Two VAM routines are used in ISD retrieval: VPAM FIND is used to locate the member, and VAM MOVE PAGE to read in the ISD. PCS also uses VISAM for off-line output in response to the DUMP command. VAM and TAM are used by the GATE routine for SYSIN and SYSOUT.

VIRTUAL MEMORY ALLOCATION

PCS uses GETMAIN to add required working storage, and FREEMAIN to release it. The CKCLS SVC is issued to determine the memory protection and privilege status associated with a virtual storage address.

DYNAMIC LOADER

PCS executes a DLINK SVC to load a referenced module. Two dynamic loader subroutines are also used by PCS. The MAPSEARCH routine is called to search the storage map for a virtual memory address. The HASHSEARCH routine is called to resolve a module name or external reference in a PCS statement. Whenever a module that was referenced symbolically in a PCS statement is unloaded, the dynamic loader enters a PCS subroutine to clear PCS tables and restore all instructions in the user's program that have been replaced with SVCs by PCS. This action guarantees that no PCS checkout or control statements will remain to interfere with execution of modules that have not been unloaded.

The task dictionary, maintained by the dynamic loader, provides PCS with a constantly updated picture of the extent of the user's program, his usage of virtual storage, and the values assigned to external symbols. A description of the task dictionary is presented in Appendix F.

COMMON AREAS

PCS processing requires a complete picture of the user's program mapping, the status of the program during execution, and the operating environment. PCS obtains this information from three basic sources: A task dictionary (TDY), internal symbol dictionary (ISD), and interrupt storage area (ISA) (Figure 2). These common areas are described in Appendix F.

OVERVIEW OF PCS PROCESSING

The processing performed by PCS for each of the commands may be conveniently discussed under two general headings: processing of immediate statements, and processing of dynamic statements. As stated earlier, a statement becomes dynamic (i.e., deferred) because of the presence of an AT command in the statement.

An overview of the processing of immediate and dynamic commands is shown in Figures 3-6. Three functional phases are distinguished: PCS Input (Phase I); PCS Input (Phase II); PCS Output (Phase III).

It should be noted, however, that this division of PCS into three phases represents a logical rather than a physical division.

PROCESSING OF IMMEDIATE COMMANDS

Phrases in immediate statements are processed individually (i.e., one at a time). Control is always returned to CA&E at the completion of processing for a phrase, unless otherwise noted.

DISPLAY and DUMP Command Processing

One or more operands are allowed in both the DISPLAY and DUMP commands. The encoded information for each of the operands is placed in a phrase list. This information consists of the starting and ending virtual storage locations to be displayed, the type of symbol that the user referenced in the operand, and the location of the appropriate dictionary entry where the symbol was found. In addition, any

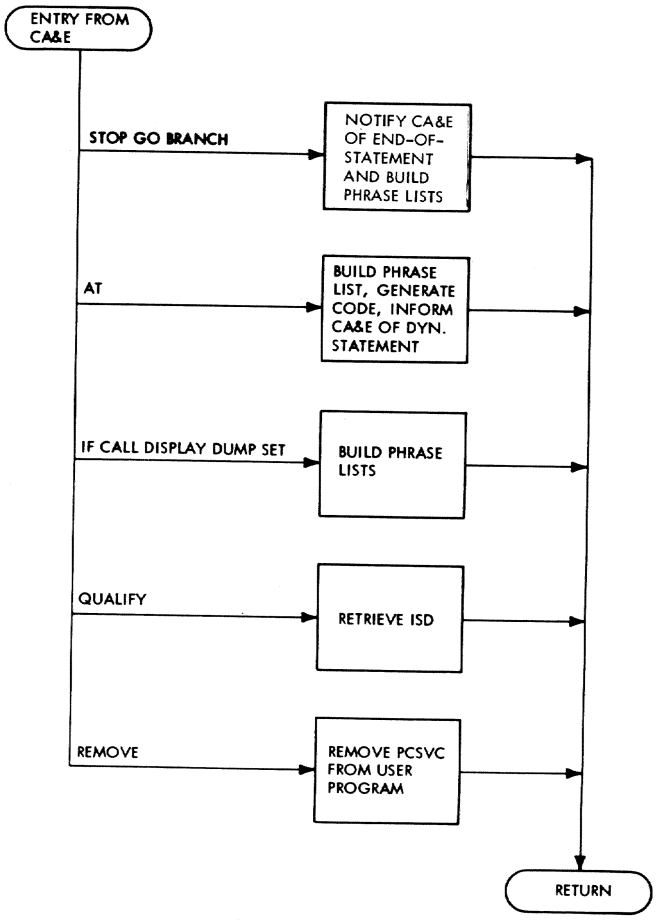


Figure 4. PCS Phase I Processing

object code that was generated (for subscripting) is located by pointers in the phrase list. The phrase list is then presented to the appropriate subroutines for displaying or dumping.

SET Command Processing

The SET command allows a data location to be set equal to an expression. The information pertaining to the data location to the left of the equal sign is tabularized into a phrase list, as in the DISPLAY command. Object code is created to compute the result of the expression to the right of the equal sign. The code is linked to via the phrase list. When the list is presented to the subroutine that performs the SET action, the object code is executed. The result of the evaluation is then returned to the SET routine, which stores it in the data location specified by the phrase list.

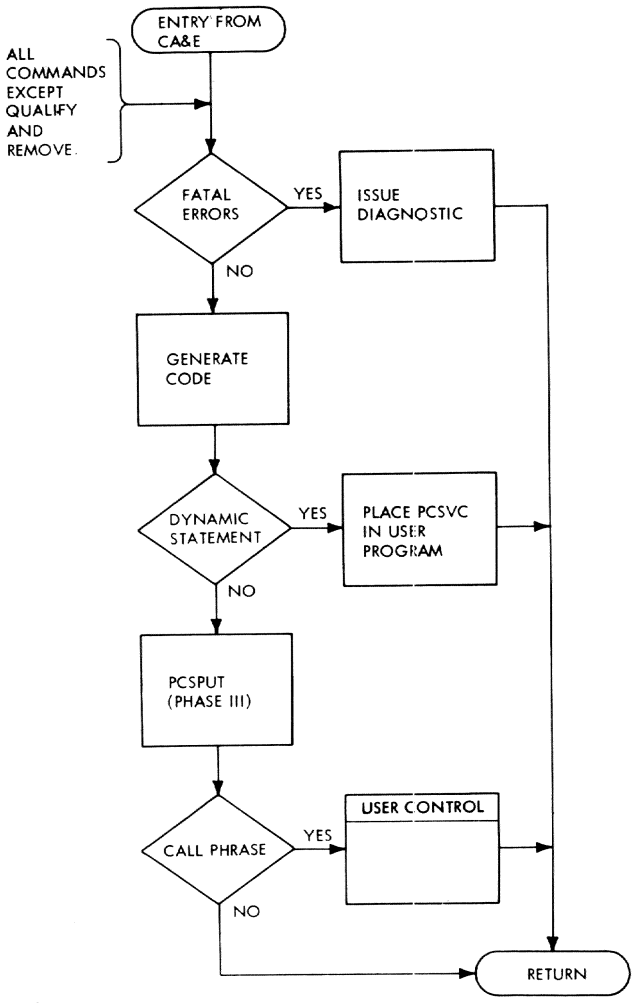


Figure 5. PCS Phase II Processing

IF Command Processing

The presence of an IF expression in the statement makes its execution conditional. PCS compiles object code to evaluate the result of the expression, executes this code, and then notifies CA&E of the result of the logical evaluation. If the condition is false, the remainder of the statement is ignored; if true, CA&E processes the next command in the statement.

QUALIFY Command Processing

The QUALIFY command allows the user to specify the name of the program module to which his internal symbols apply. Processing of the directive includes locating the internal symbol dictionary for the module and storing the necessary information into its internal tables.

REMOVE Command Processing

The REMOVE command allows the user to deactivate selected dynamic statements permanently. PCS processing consists of delinking and removing the appropriate table entries for the statements and of removing any PCSVC's present.

CALL Command Processing

The CALL command initiates program execution. If a module has not been loaded, PCS will request the dynamic loader to load it. The VPSW is then modified to start execution at the module entry point, and the contents of the user's linkage registers are set so that a type-I linkage can be achieved. Control is given to the User Control Routine for this.

STOP Command Processing

The STOP command in an immediate statement causes the user to be notified of the current status of his program. CA&E is notified that an end-of-line condition is met.

BRANCH Command Processing

The BRANCH command causes the VPSW (which contains the location in the user's program from which execution will start or be resumed) to be altered to an operand specified in the command. CA&E is notified of an end of statement.

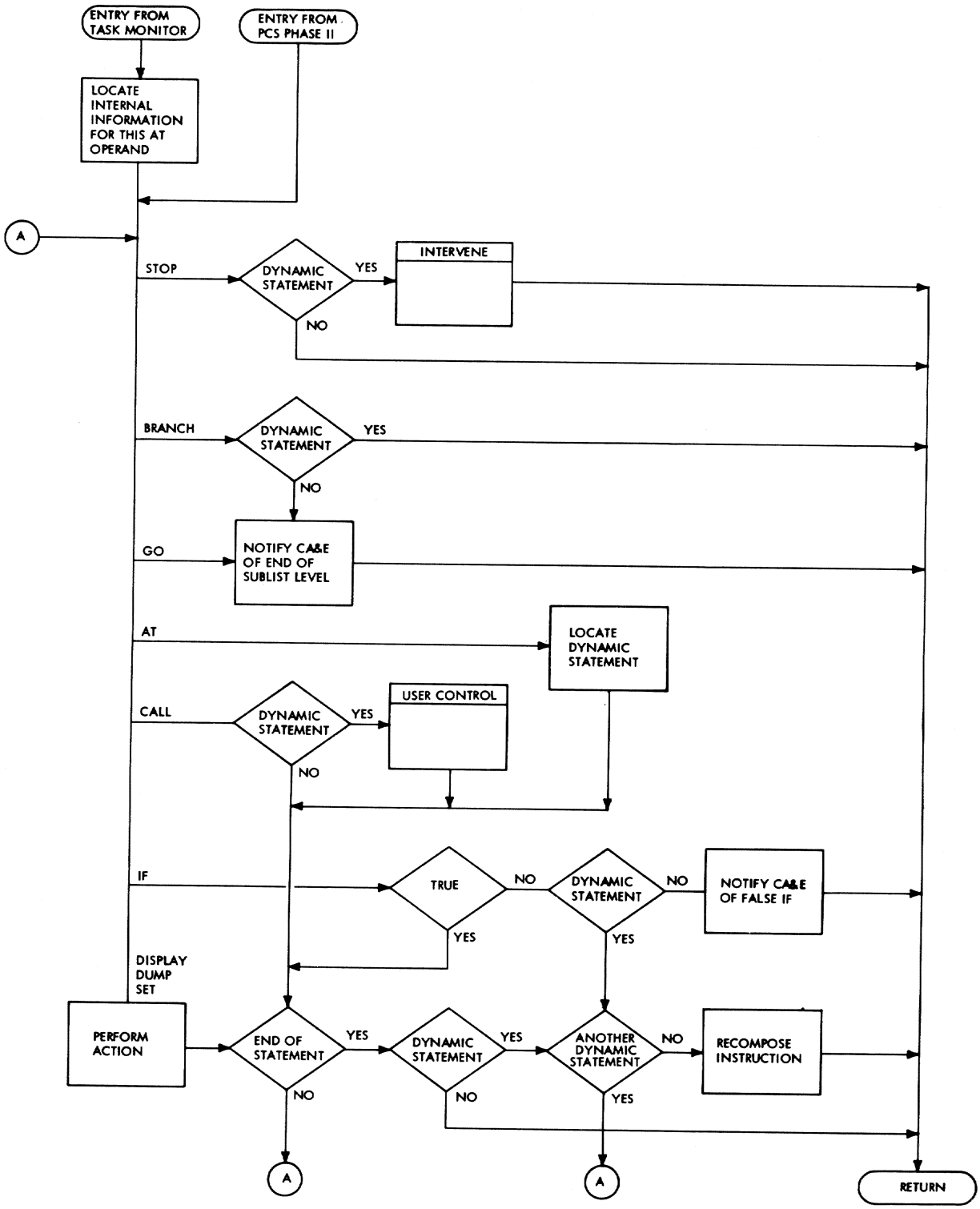


Figure 6. PCS Phase III Processing

GO Command Processing

When a GO command is entered, PCS notifies CA&E of an end of line. Since the location of the VPSW was not altered, execution will start or resume at the current address.

PROCESSING OF DYNAMIC COMMANDS

Dynamic commands are commands whose execution has been deferred until a specified location in the user's program is reached during execution. This is accomplished by means of the AT command.

AT Dynamic Command Processing

PCS implements the AT command by inserting a PCSVC into the user's program at the location specified in the command operand. The user's instruction at that location is saved in internal tables. PCS then notifies CA&E that the statement is dynamic.

When the PCSVC is executed in the user's program, the task monitor recognizes it and enters PCS for processing (Figure 7). PCS searches its internal tables for the information pertaining to the SVC at that location. All the processing is then done for the actions requested at that event, as described above in the paragraphs on processing of immediate commands, except as indicated below.

IF Dynamic Command Processing

The condition is evaluated by executing the code generated during the immediate command processing phase. If the result is false, the next statement effective at this location is processed. If the result is true, the remaining phrases in the current statement are performed.

STOP Dynamic Command Processing

The user is notified of the location of his program and control is given to the Intervene system routine to halt the user's program and to place the task in the command mode.

BRANCH Dynamic Command Processing

The VPSW in the ISA is modified and control is returned to the task monitor to resume program execution.

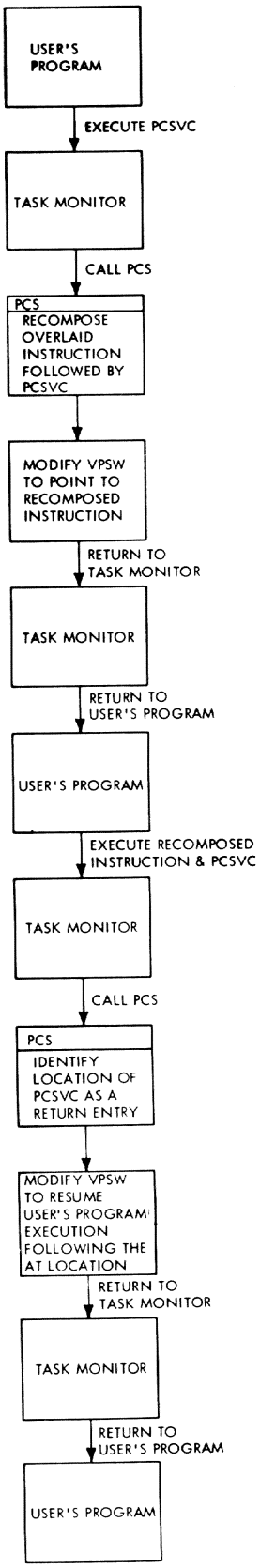


Figure 7. Sequence of Events of AT Statement Processing

CALL Dynamic Command Processing

Control is given to the User Control system routine which initiates execution of the called program. When control returns from the program, the remaining actions in the dynamic statement are performed.

Terminal Processing of Dynamic Statements

Prior to resumption of the user's program, the instruction that was overlaid with the PCSVC must be executed (Figure 7). This is done by recomposing the instruction into a PCS work area. After the recomposed instruction is executed, return is made to the user's program at the next sequential instruction. To do this, a second PCSVC is placed following the recomposed instruction so that PCS will again be entered immediately after its execution. PCS recognizes this entry as a return-entry and modifies the VPSW to point to the next sequential instruction in the user's program. Control is then passed to the task monitor and normal program execution continues.

The Introduction to this manual presented an overview of PCS functions, without specific reference to the program internals. The present section relates these functions to specific routines. Brief descriptions are given of each function, the routines that perform these functions, the calling relationship between routines, and the internal communication tables that are formed and referenced by them. This discussion is prefaced by a brief account of the logical organization of PCS.

PCS PHASE DESCRIPTIONS

Processing of PCS statements is performed in three logical phases, designated as PCS Input (Phase I), PCS Input (Phase II), and PCS Output (Phase III).

The functions for each logical phase are performed by a collection of subroutines under the direction of a control subroutine for that phase. The phases, their functions, and the control routines, are described briefly below.

It should be understood that there is no one-to-one correspondence between these logical phases and physical PCS assembly modules; many PCS routines are employed in more than one logical phase.

PCS Phase I

PCS Input (Phase I) performs initial processing of PCS statements. Commands and their operands are evaluated, checked for errors, and then formed into phrase lists for further processing by Phases II and III. Control is exercised by one of nine Phase I control routines; CA&E passes control to the appropriate routine depending on the source phrase currently being processed.

PCS Phase II

PCS Input (Phase II) performs final input processing of PCS statements. If any warning diagnostics were issued in Phase I, Phase II prompts the user to accept the system's interpretation. In nonconversational mode, warning diagnostics cause the statement to be ignored.

If the user accepts the statements as interpreted by the system, or if there are no diagnostics, Phase II proceeds to generate all object code necessary to evaluate SET and IF commands, and subscripted variables. The generated code is linked to PCS tables. If the statement is dynamic, all PCSVCs called for by AT commands are planted in the user's program. If the statement is immediate, Phase II calls PCS Output (Phase III) to perform the actions specified.

CA&E initiates Phase II processing by a call to PHASE2 (CZANA): this routine controls the processing described above. If the user does not accept the statement or if any fatal diagnostics are issued, the above processing does not occur and PHASE2 returns control to CA&E.

PCS Phase III

PCS Output (Phase III) performs the actions specified by PCS internal tables built during Phase I and II. PCS Output is entered either from the task monitor, as the result of a PCSVC having been executed in the user's program, or, in the case of immediate statements, directly from the Phase II control routine, CZANA. Entry is made to PCSPUT (CZAPB), the control routine for Phase III processing.

PCS Output (Phase III) calls a DISPLAY/DUMP component whenever the DISPLAY, DUMP, or SET functions are to be performed.

Following the completion of Phase III processing, PCSPUT (CZAPB) returns control to the caller.

PCS ROUTINES AND COMMUNICATION TABLES

This sub-section relates PCS functions to specific routines and internal communication tables and areas. The organization of this material

reflects the division of PCS functions into three logical phases. Overview flowcharts are given of the control routines for each of the phases. Nesting charts are given showing the calling relationship between the routines of each phase. Tables specifying routine calling conditions are given in Appendix C.

For detailed information concerning a specific routine, the reader should consult the individual routine description in Section 3, or the flowchart in Section 4. These sections are ordered by module ID. The reader will find detailed descriptions of external and internal communication tables in Appendix F.

Note: Routine designations appearing in this document take the CZ... form. By convention, the CF designation is intended to denote routines referenced exclusively from within the same assembly module; the CZ designation denotes routines referenced from outside the assembly module. However, since this convention has not been strictly followed, the CZ code is used throughout this document to designate both cases. The correct designation for internally referenced routines can be determined by checking the listings.

PCS INPUT (PHASE I)

PCS Input (Phase I) routines can be divided functionally into five catagories:

- Control routines
- Source list scanner
- Expression evaluators
- Operand evaluators
- Diagnostic routine

Control Routines

When CA&E recognizes a phrase requiring PCS processing, control is passed to the appropriate Phase I control routine (Figure 8). These are: SET (CZAMA), BRANCH (CZAMB), STOP (CZAMC1), GO (CZAMC2), DISPLAY (CZAMD1), DUMP (CZAMD2), IF (CZAME), AT (CZAMF), explicit CALL (CZAMG1), and implicit CALL (CZAMG2). All of the above routines control the evaluation of operands into encoded form and the insertion of these operands into phrase lists for further processing by Phases II and III.

Control for the processing of the QUALIFY and REMOVE commands is provided by QUALIFY (CZAMR) and REMOVE (CZAMS). QUALIFY (CZAMR) locates the ISDMAP entry for the program module to be used in the implicit qualification of internal symbols. REMOVE (CZAMS) performs permanent cancellation of specified dynamic statements. Phase I provides the entire processing required by the QUALIFY and REMOVE commands.

UNLOAD (CZAMT) is entered from the dynamic loader when a module referenced in PCS statements is unloaded by the user. The routine provides all the processing needed to release PCS working storage and provides for restoring the user's instructions that were overlaid with PCSVCs.

The remaining routine categories represent those called directly or indirectly by Phase I control routines. The nesting relationships for Phase I routines can be seen in Figure 9. Their calling conditions are defined in Appendix C.

Source List Scanning Routine

SCANFLD (CZAMQ) scans the source list and forms a source list item (SLITEM). This item and the other internal reference items mentioned in the succeeding paragraphs are identified immediately afterwards.

Expression Evaluator Subroutines

EXPSCAN (CZAMH) evaluates an expression and forms a Polish string (POLISH). The subscript/offset scan routine, SUBPOL (CZAMJ) evaluates a subscript/offset expression and forms a Polish string.

Operand Evaluator Subroutines

DATALOC forms a data location item (LOCITEM). DATAFLD (CZAMI) forms a data field item (FLDITEM) which may consist of a single data location item or two combined data location items, forming a range. VALSYM (CZAOB) forms an internal symbol data location item. EXTERNAL (CZAMO) forms an external symbol data location item. It also forms a data location item for an

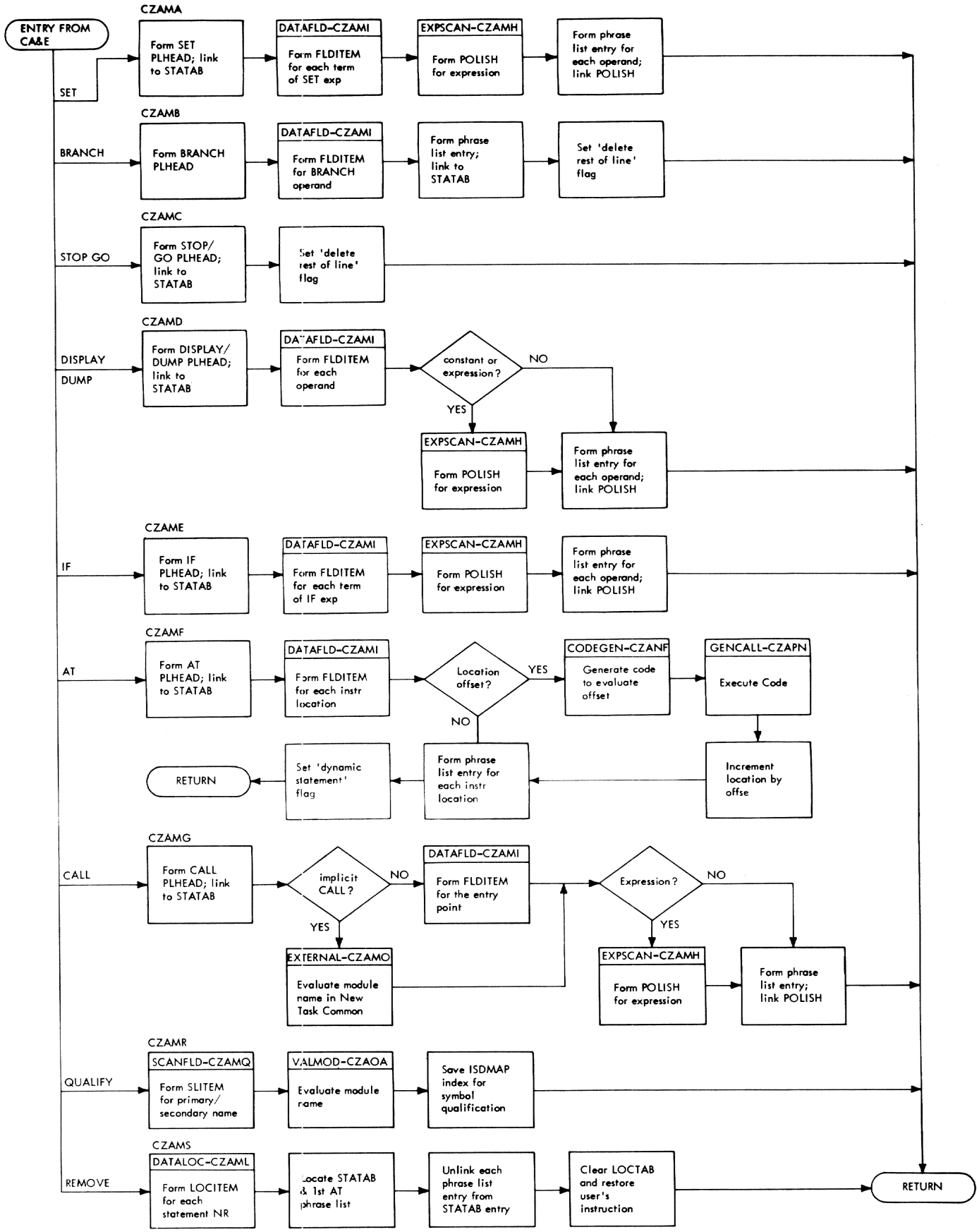


Figure 8. Phase I Control Routine

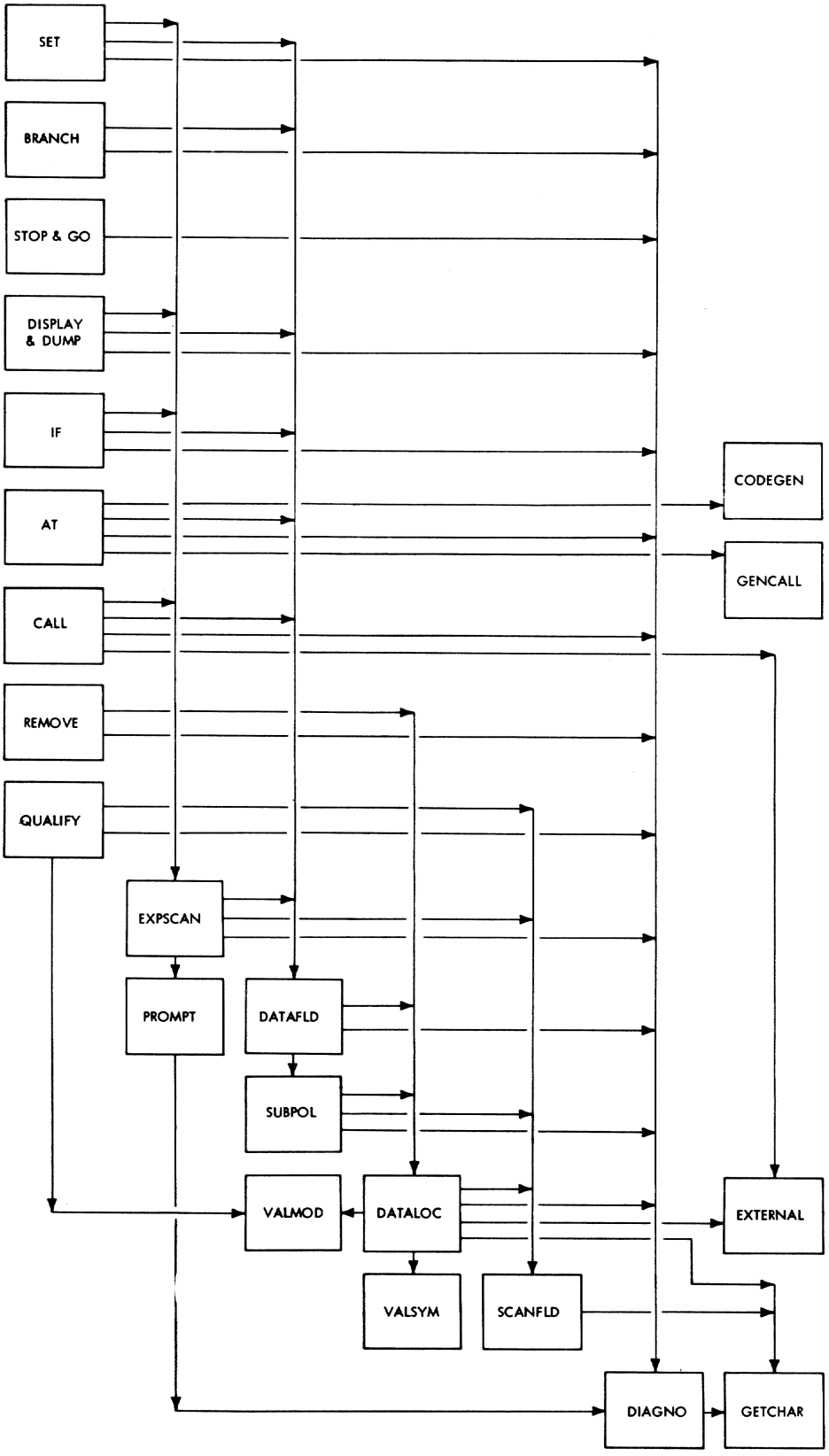


Figure 9. Phase I Nesting Chart

undefined command variable or a defined command variable. VALMOD (CZAOA) evaluates a module name and locates internal symbol dictionaries (ISD) for the module.

Diagnostic Routine

Each Phase I routine, while performing the various error checks, can form diagnostic codes which subsequently will cause diagnostic messages to be issued.

The diagnostic routine DIAGNO (CZANW) selects the text of the message based on the code. The severity of the error is also decided by the diagnostic code. The fatality code of the entire statement is based on the most severe diagnostic issued.

PHASE I REFERENCES TO INTERNAL TABLES

PCS processing builds entries or makes references to the following internal communication tables and areas:

- Statement Table (STATAB)
- Phrase List (PLHEAD)
- Source List Item (SLITEM)
- Identified Source List Item (PQNITEM, SQNITEM, SYMITEM, SUBITEM)
- Data Location Item (LOCITEM)
- Data Field Item (FLDITEM)
- Polish String (POLISH)
- Internal Symbol Dictionary Map (ISDMAP)
- Location Table (LOCTAB)
- Display List (DISPLIST)

These areas and their use by PCS are described below. The description is preceded by a summary review of source list processing by CA&E.

Source List

CA&E initializes a source list each time a user logs on. As CA&E obtains a user's command, it places the command in a level-one sublist and calls the appropriate routine in PCS Input (Phase I). Should the user's program be executed during this session and should the execution of that program be interrupted, for example, by the issuance of a new command, CA&E will create a nested sublist (a non-level-one sublist) for the new command. The new sublist acts in the same way as a level-one sublist, initiating a user's program, etc. When all nested, non-level-one sublists are processed, processing of the level-one sublist is resumed.

Statement Table (STATAB)

A STATAB entry is created by the appropriate Phase I control routine for each PCS statement to be processed. The entry points to the first phrase list header (PLHEAD) for the statement. This phrase list header identifies the first PCS command in the statement.

If the statement is immediate, the STATAB entry is placed in the PCS PSECT. If the statement is dynamic, the immediate STATAB entry is assigned temporary storage in the last STATAB page. Phase II later assigns this storage permanently.

Phase III will use STATAB to locate the phrase list in processing both immediate and dynamic statements.

Phrase List (PLHEAD)

In processing PCS statements, Phase I control subroutines build a phrase list for each PCS command. This list consists of a phrase list header (PLHEAD), identifying the command by type, and a phrase list entry for each of the command operands. The format of each phrase list entry is determined by the syntax used to express the operand and the process used to evaluate it.

Source List Item (SLITEM)

Characters are extracted from the source list by GETCHAR and are stored as a continuous string in the buffer in PCS PSECT. The SCANFLD (CZAMQ) subroutine scans this string and forms a source list item (SLITEM) for each string. SLITEM contains the location and length of the string, the character type (numeric or alphameric), and the string delimiter found.

Identified Source List Item

The identification and delimination of the source list item are used to

determine the format of the data location (LOCITEM). During this process of identifying the data location, DATLOC moves the source list item previously formed to one of four identified source list items: PQNITEM, SQNITEM, SYMITEM, and SUBITEM. PQNITEM and SQNITEM specify primary and secondary qualifying module names. SYMITEM specifies symbol names. SYMITEM and SUBITEM specify statement numbers and subscripts. All four identified source list items are used to express a floating point constant. All other data locations are specified by SYSITEM.

Subsequent processing of the identified source list items varies according to the type of LOCITEM identified.

Data Location Item (LOCITEM)

A data location item (LOCITEM) is formed by Phase I from the identified source list items. LOCITEM identifies the data location as an array, internal symbol, internal symbol with offset, statement number, statement number with offset, single precision constant, double precision constant, etc. Depending on the data format used, several source list items may be required to identify the data location item completely.

When a LOCITEM has been identified and validated by a context check, control is passed to the appropriate Phase I routine to perform the conversion and/or evaluation.

Data Field Item (FLDITEM)

A Data Field Item (FLDITEM) is formed from the LOCITEM. If subscript/offset notation is used in a LOCITEM, Phase I forms a Polish string for the subscript/offset expression. If range notation is used, a second LOCITEM and Polish string is formed and incorporated into FLDITEM. The DATAFLD (CZAMI) routine controls formation of FLDITEM.

Polish String (POLISH)

If the operand is an expression, the Phase I control routine calls EXPSCAN (CZAMH) to form a Polish string. Entries in a Polish string consist of a header entry for the expression, an entry for each operand and operator in the expression and a trailer entry to control the processing of the Polish string by Phase II. If an operand is subscripted or offset, SUBPOL (CZAMJ) is called to evaluate it, and the operands and operators of the subscript/offset Polish string are included in the Polish string for the expression.

A description of Polish string formation is given in the module description for EXPSCAN (CZAMH). An example of a Polish string processing is given in the module description for CODEGEN (CZANF). Further discussions of Polish strings are in Appendix F.

Internal Symbol Dictionary Map (ISDMAP)

The ISDMAP, generated by Phase I, contains an entry pointing to the address of each ISD loaded. If the ISD is a link-edited ISD, additional ISDMAP entries are made for each assembled or compiled module.

Location Table (LOCTAB) and Display List (DISPLIST)

In addition to the tables and areas described above, PCS processing includes references to a location table (LOCTAB) and a display list (DISPLIST). Entries to LOCTAB are created by Phase II and are described in the discussion for that phase. Phase I references LOCTAB in processing REMOVE and UNLOAD phrases. DISPLIST entries are created by Phase III and are described in the discussion for that phase. Phase I does not reference DISPLIST.

Final Phase I Processing

When the format for the phrase list entry has been fully determined, Phase I control routines insert entries for each operand into the phrase list behind the phrase list header and establish linkage to any POLISH string formed in the process.

When the current phrase list has been completely formed, control is returned to CA&E.

PCS INPUT (PHASE II)

Phase II subroutines can be divided functionally into four catagories:

- Control Routine
- Code Generation Routines
- Table Scan Routine
- Diagnostic Routines

Control Routine

When Phase I returns control to CA&E, CA&E calls CZANA (PHASE2), the control routine for Phase II processing (see Figure 10).

If fatal diagnostics were issued in Phase I, CZANA calls DIAGNO (CZANW) to issue a diagnostic and then informs CA&E to delete the rest of the source line. For less-than-fatal diagnostics, CZANA calls PROMPT (CZANX) to write a diagnostic and solicit user approval of the system's interpretation. (In non-conversational mode, the statement is ignored).

If the user's approval is obtained, CZANA proceeds to locate and initialize the STATAB entry and to locate the associated phrase list. Each entry in the phrase list is then inspected to determine if the phrase list entry contains a pointer to a polish string. If so, CZANA calls CODEGEN (CZANF) to generate code to evaluate the polish string. The address of the generated code then overlays the address of the Polish string in the phrase list entry.

If the statement is dynamic, the AT phrase is processed after the generation of all necessary code. A LOCTAB entry (described below) is formed for each entry in the AT phrase list and a PCSVC is stored in the user's program. The LOCTAB entry serves to link the PCSVC address with the first entry in the statement table (STATAB) associated with the address. If a previous LOCTAB entry already exists for the location specified by the AT phrase list entry, as the result of an earlier dynamic statement, the current STATAB entry is linked to the last STATAB entry for the location.

If the statement is immediate, CZANA calls Phase III to perform the necessary actions.

Code-Generation Routines

CODEGEN (CZANF) is the control routine for all object-code generation. CFANF processes the polish string, and based on the type of entries, calls the appropriate routine to combine two operands. If the operator is part of a dimension string, SUBGEN (CZANG) is called to generate the code.

If the operation is between two non-dimensioned variables or between a non-dimensioned variable and a constant, the code to perform the operation is generated by OPGEN (CZANI). If both operands are constants, COMCON (CZANH) is called to combine them. LOADOP (CZANT) generates code to load an operand. GETBASE (CZANV) assigns a base register for referencing an operand and generates code to load the base register when necessary. GETREG (CZAOD) assigns registers required for generated code.

Table Scan Routine

If the statement is dynamic, CZANA calls FINDLOC (CZAPC) to scan the location table (LOCTAB) for an entry which matches the specified AT location. If an available (i.e., non-matching) entry is found, its address is returned and CZANA creates a LOCTAB entry to link the PCSVC address and the STATAB entry. If a matching entry is found, CZAPC returns a matching code, signifying that a PCSVC already exists for the location. CZANA will then link the matching LOCTAB entry to the current STATAB entry.

Diagnostic Routines

DIAGNO (CZANW) forms and issues diagnostics. Diagnostic levels are:

1. Null - These are informational only and do not result in prompting or rejection.

2. Warning - In conversational mode, these are informational and cause the user to be prompted for acceptance of system's interpretation. In non-conversational mode, warning diagnostics cause the statement to be ignored.

3. Operand Fatal - These indicate that an operand is being ignored. They cause prompting or rejection.

4. Statement Fatal - These diagnostics cause the entire statement to be rejected.

PROMPT (CZANZ) issues prompting messages as a result of a diagnostic, and solicits a user's response.

PHASE II REFERENCES TO INTERNAL TABLES

Phase II builds entries or makes references to the following internal

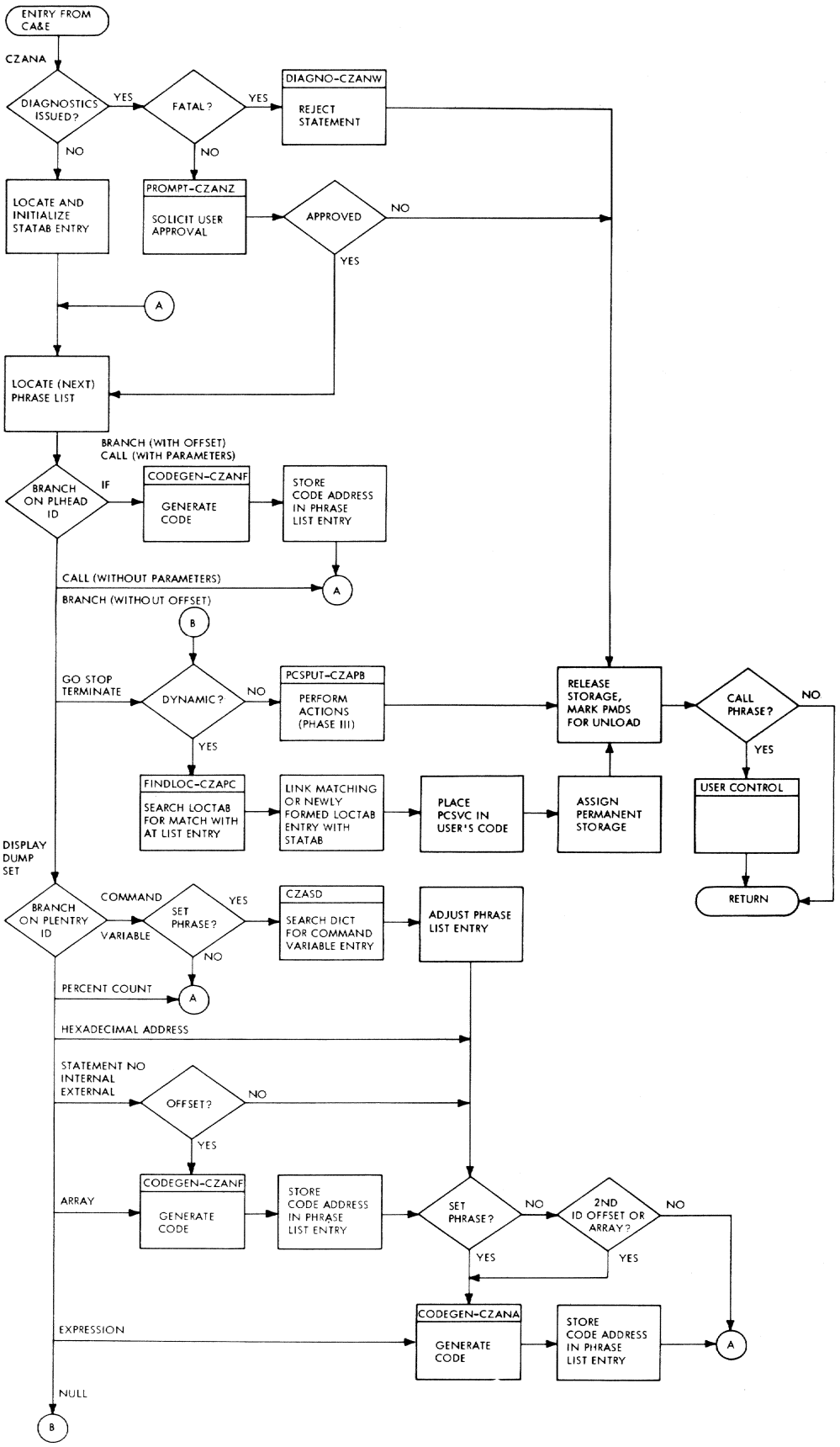


Figure 10. Phase II Control Routine

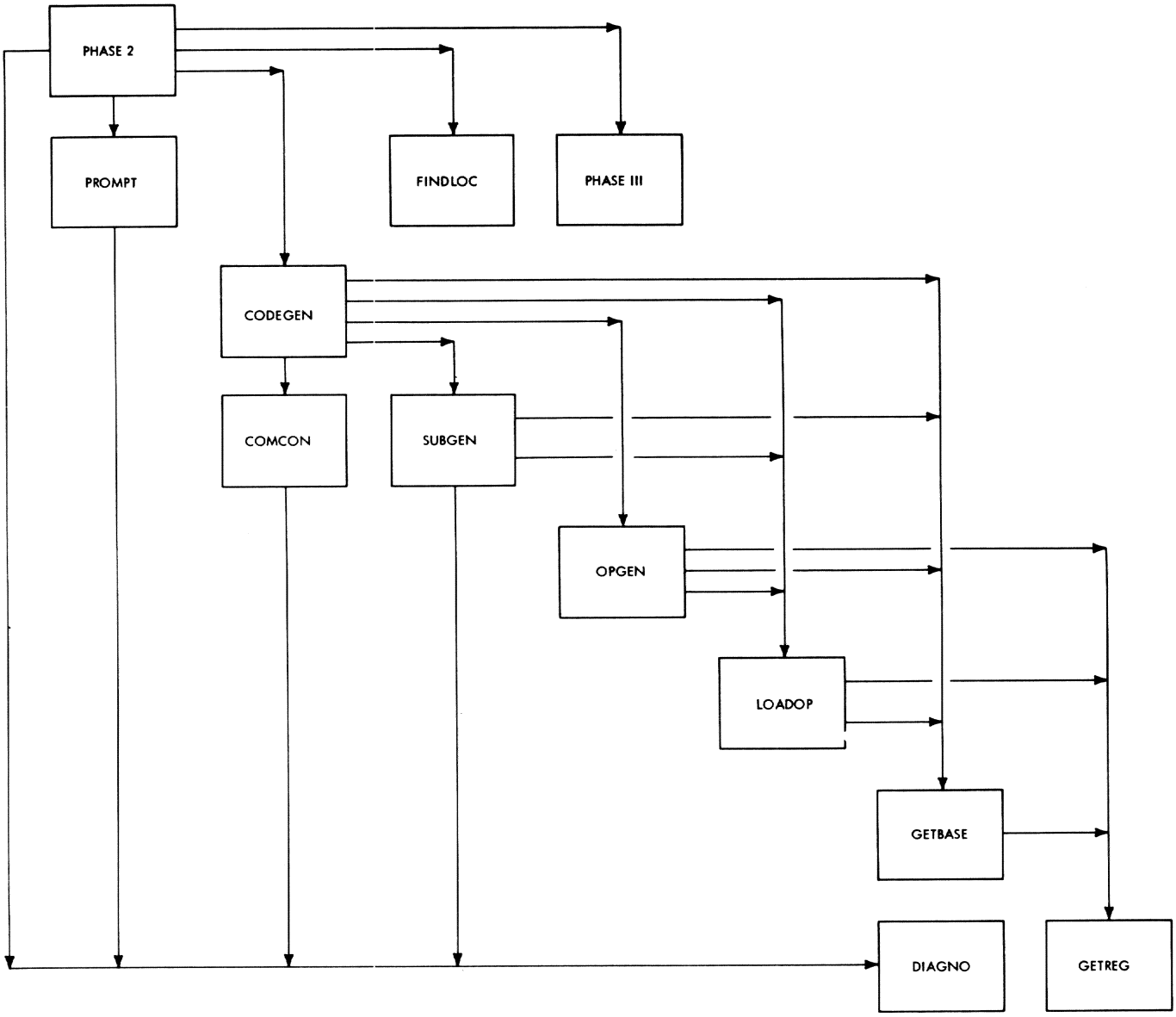


Figure 11. Phase II Nesting Chart

communication tables and areas. Except for LOCTAB, which is described below, these areas are described in the discussion for Phase I.

Statement Table (STATAB)

Phase II locates and initializes the STATAB entry. If the phrase being processed is dynamic, Phase II assigns permanent storage to the immediate STATAB entry.

Phrase List (PLHEAD)

Phase II locates the phrase list(s) formed by Phase I. Phase II then inspects the identification of each phrase list header (denoting type of command) and phrase list entry (denoting type of operand) and performs further processing depending on the identification found.

Polish String (POLISH)

If the phrase list entry contains a pointer to a polish string, Phase II generates code to evaluate the polish string. The address of the generated code then overlays the address of the polish string in the phrase list entry.

Location Table (LOCTAB)

This table links PCSVC interrupts to the deferred PCS statement. Entries

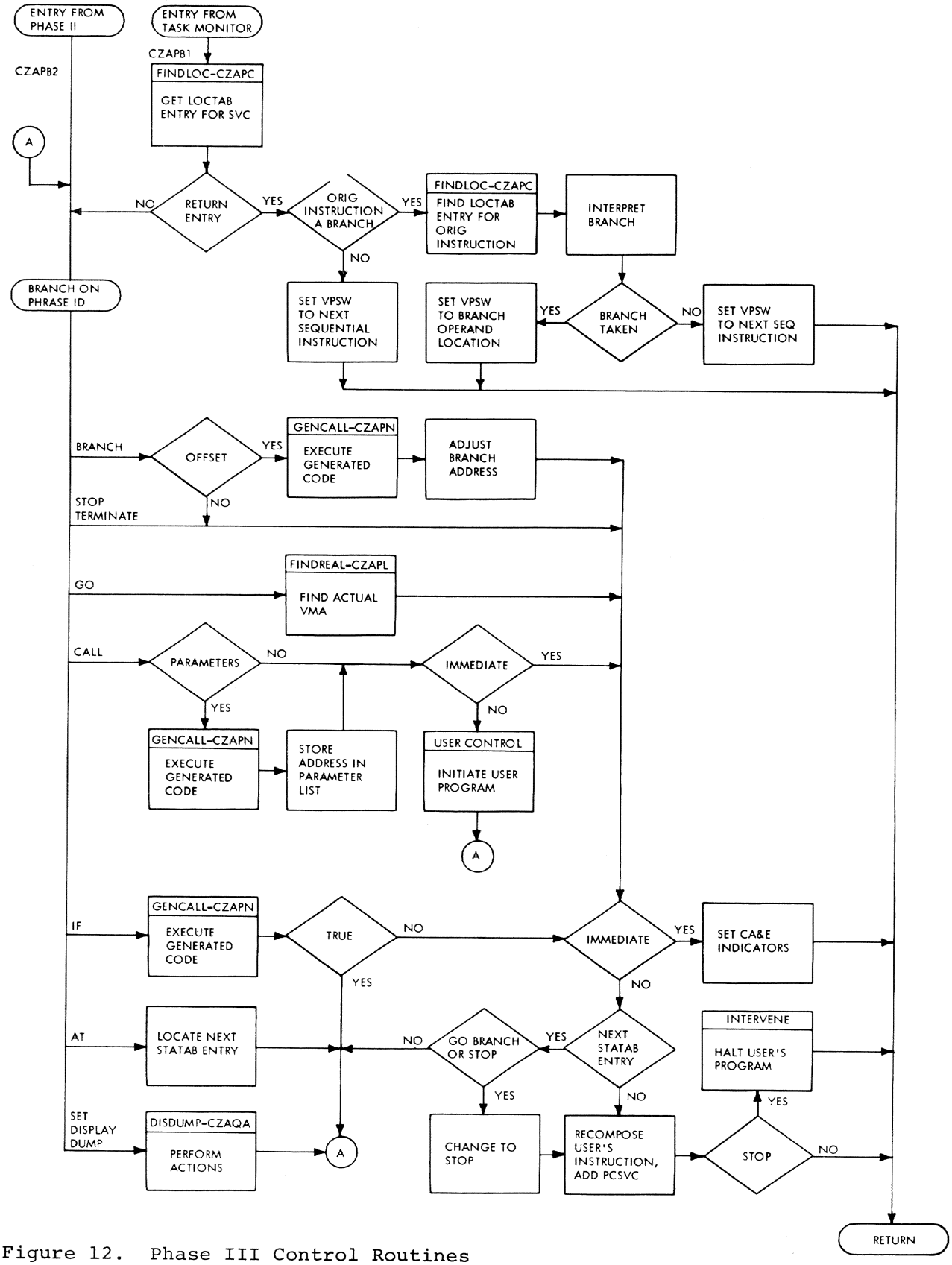


Figure 12. Phase III Control Routines

in the table link the PCSVC address with the first entry in the statement table (STATAB) associated with this location. Phase II creates LOCTAB entries for each PCSVC stored in a user's program, while Phase III creates LOCTAB entries for each PCSVC that follows a recomposed instruction. The relative position of the entry in the table is determined by a hashing algorithm, based on the address of the PCSVC.

Phase I references this table in processing REMOVE and UNLOAD phrases. Phase III, in processing the PSCVC, locates the LOCTAB entry by means of the hashing algorithm mentioned above.

PCS OUTPUT (PHASE III)

Phase III routines can be grouped into three catagories:

- Control routine
- Processing routines
- Diagnostic routines

These routines are identified and described below.

Phase III also contains a DISPLAY/DUMP component which is entered whenever the DISPLAY, DUMP or SET functions are to be performed. This component is described following the discussion of the main part of Phase III processing.

Control Routine

Phase III control is exercised by PCSPUT (CZAPB). This routine has two entry points; it is entered from Phase II to process an immediate statement, and from the task monitor when processing a dynamic statement.

To determine which type of dynamic statement entry caused the interrupt, CZAPB, upon receiving control from the task monitor, calls FINDLOC (CZAPC) to locate the LOCTAB entry for the PCSVC. If the LOCTAB entry is not a RETURN entry, it signifies that the interrupt was caused by a PCSVC which was inserted into the user's program during Phase II execution. In this case, CZAPB processes the dynamic statement as if it were an immediate statement. If the entry is a RETURN ENTRY, it signifies that the interrupt was caused by the PCSVC which was placed immediately after the recomposed user instruction earlier during Phase III processing. In this case, CZAPB processing consists of modifying the VPSW to point to either the address of the next sequential instruction, or to the branch address, in cases where the original instruction could result in a branch.

In processing immediate statements and non-RETURN dynamic statements, CZAPB locates the STATAB entry and its associated phrase list. A branch is then made, depending on the phrase list identification, to one of the following processes.

AT: Each entry in the AT list is checked to see if the entry is for the current interrupt address. If an entry is found for the current interrupt address, the address of the next STATAB entry is located. Processing continues with the identification of the next phrase list.

IF: GENCALL (CZAPN) is called to execute the code generated in Phase II for evaluation of the IF expression. If the result is true, processing continues with the identification of the next phrase list. If the result is false, and a dynamic statement is in process, processing continues with the next STATAB entry. CA&E is notified of a false immediate IF phrase.

DISPLAY, DUMP, AND SET: DISPDUMP (CZAQA) is called to process the phrase list. Processing continues with the identification of the next phrase list.

STOP: The STOP indicator is set. Processing continues with the next STATAB entry.

GO: The user is notified of the symbolic instruction where program execution is resumed. The BRANCH/GO indicator is set. Processing continues with the next STATAB entry.

BRANCH: If the branch address is offset, GENCALL (CZAPN) is called to execute the code generated in Phase II. The result is added to the branch address. The user is notified of the symbolic location where program execution is resumed. The BRANCH/GO indicator is set. Processing continues with the next STATAB entry.

CALL: If a parameter list is to be constructed, GENCALL (CZAPN) is called to execute generated code to evaluate each parameter. The address of each parameter is stored in the parameter list. The V-con, R-con, and the parameter list are inserted in the source list. If the statement is immediate, processing continues with the next STATAB entry. If the statement is dynamic, the User Control routine is called to initiate program execution. When control returns, processing continues with the next phrase list.

TERMINATE: Processing continues with the next STATAB entry and for immediate statements, control is returned to the caller. CA&E is notified of an end-of-level if a BRANCH or GO phrase was processed.

For dynamic statements, the next STATAB entry is processed as described above for each command. When all STATAB entries have been processed, the user's instruction is recomposed. This is followed by a PCSVC. The LOCTAB entry for the PCSVC is designated a RETURN entry. Control then returns to the caller, unless the user program has been halted.

The remaining categories of routines represent those called directly or indirectly by CZAPB to accomplish specific tasks. The nesting relationship for Phase III routines can be seen in Figure 13. Their calling conditions are defined in Appendix C.

PROCESSING ROUTINES: GENCALL (CZAPN) is entered with a pointer to the code generated in Phase II. The subroutine executes the code and stores the result. FINDREAL (CZAPL) determines if the VMA with which it is entered contains a recomposed instruction. If so, the caller is returned to the VMA of the original, overlaid instruction. SYMGEN (CZAPG) converts a VMA to symbolic form for display purposes. SAVIX (CZAPK) recomposes the machine instruction in the user's program and follows it with an ENTER PCSVC. FINDLOC (CZAPC) hashes the VMA of a PCSVC and finds the matching LOCTAB entry.

Diagnostic Routine

LINE (CZAPH) outputs confirmation and diagnostic messages to the user. A description of the formats for these messages is provided following the discussion of Phase III.

PHASE III DISPLAY/DUMP COMPONENT

The DISPLAY/DUMP component of Phase III is called by CZAPB whenever DISPLAY, DUMP, or SET functions are to

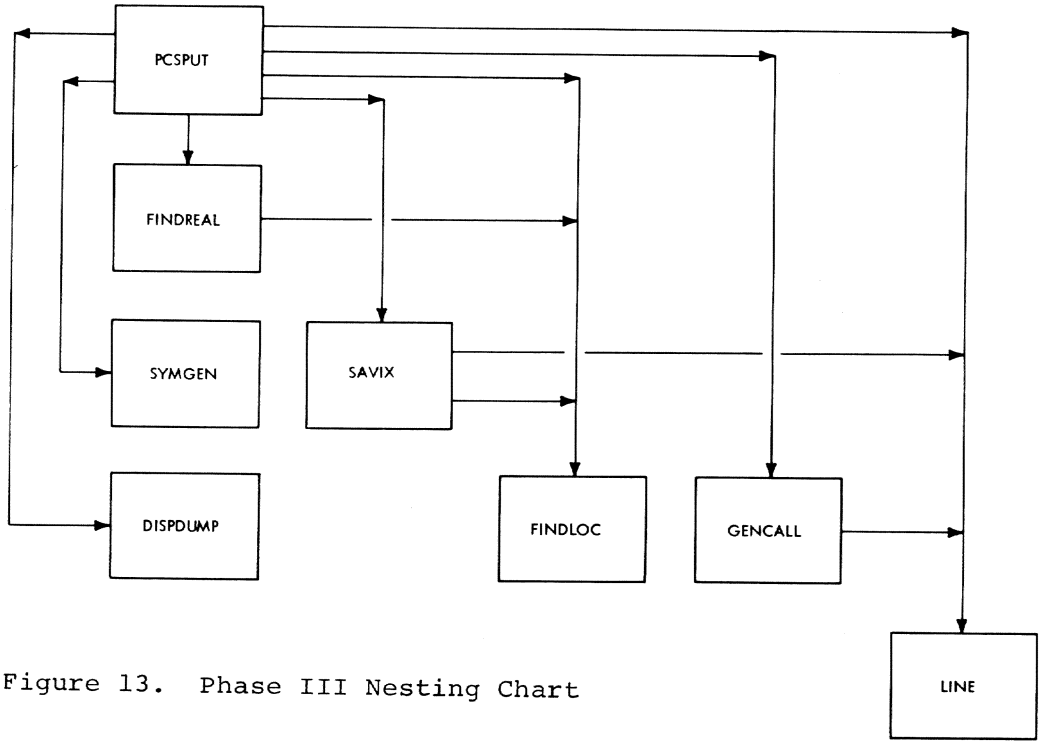


Figure 13. Phase III Nesting Chart

be performed (see Figure 14). CZAPB passes the address of the first phrase list as an argument.

Processing of the DISPLAY/DUMP component is controlled by DISPDUMP (CZAQA). The following functions are performed:

1. Picks up successive item references from the phrase list, obtains the address and attributes of each item, converts the contents of each item according to its attributes and places it in an area for output.

2. If DISPLAY is specified, transmits values of items in a list to the user's SYSOUT.

3. If DUMP is specified, generates the same values onto a data set referenced as PCSOUT.

4. Modifies and displays the contents of a data location referenced by a SET command.

Only one display list (described below) is processed for each entry into DISPDUMP. Each entry in the phrase list is processed and two display list items (DISPLIST) are formed. The DISPLIST item defines the attribute of the phrase list entry data. These attributes determine the output format.

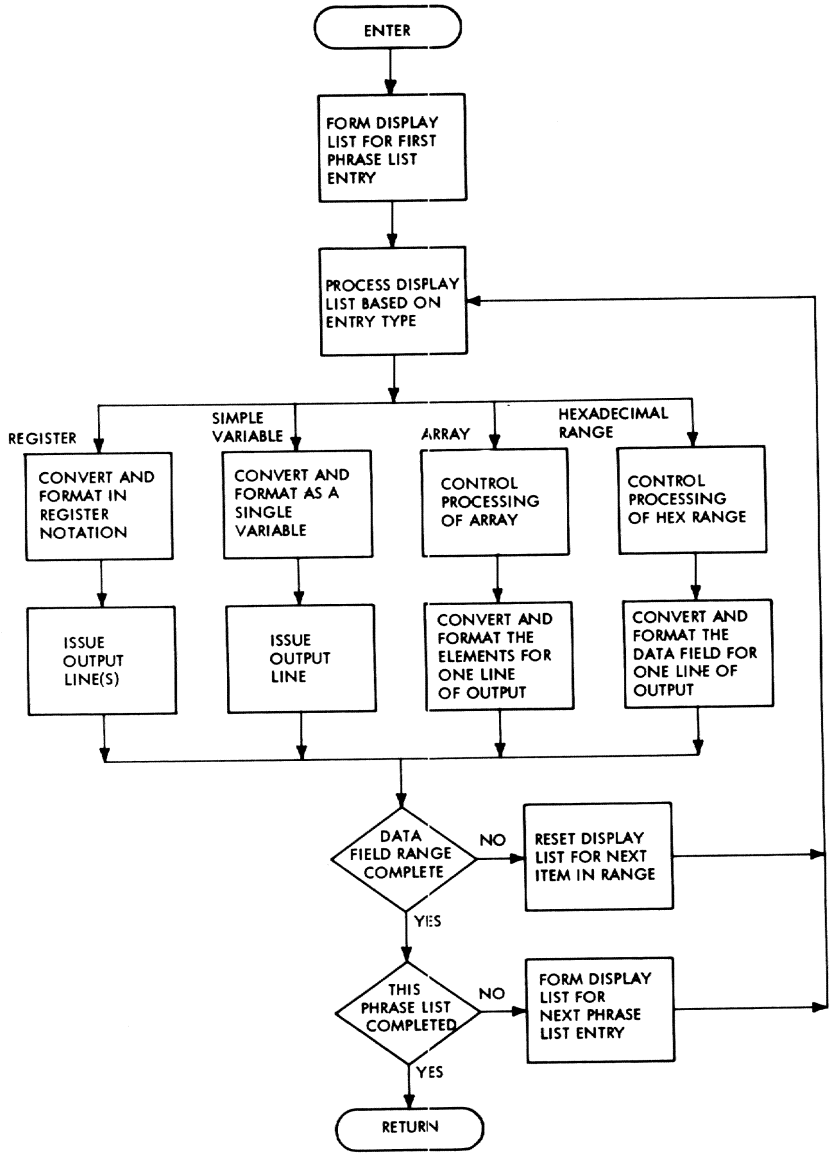


Figure 14. Display/Dump Control Routine

The formats of the data displayed corresponds to the way the data field was specified in the PCS statement. Twelve specifications are defined, each with a unique IDENT value and entry format in the DISPLAY/DUMP phrase list. These are:

1. General rigister
2. Single precision register
3. Double precision register
4. % count
5. Internal symbol
6. FORTRAN statement number
7. Subscripted array
8. External symbol
9. Hexadecimal address
10. Expression
11. Command Variable
12. 5, 6, 8, and 9, above, with offset

A complete description of DISPLAY/DUMP output formats is provided in Appendix G.

PHASE III REFERENCES TO INTERNAL COMMUNICATION TABLES

Phase III builds entries to or makes references to the following internal communication tables and areas:

Statement Table (STATAB)

CZAPB references STATAB to locate the phrase list for the current statement. In immediate statement entries to Phase III, the location of the STATAB entry is supplied as a parameter. In non-RETURN dynamic entries, the location of the STATAB entry is picked up from the LOCTAB entry.

Location Table (LOCTAB)

In dynamic statement entries, CZAPB references the LOCTAB entry for the PCSVC to determine whether the entry is a RETURN or non-RETURN entry. In subsequent processing of non-RETURN dynamic statements, LOCTAB is referenced to locate the address of the overlaid instruction.

Phrase List (PLHEAD)

CZAPB obtains the command identification from the phrase list header (PLHEAD); the path of processing is determined by the identification. DISPDUMP uses the phrase list entry identification of the operand type to form a display list (DISPLIST).

ISD Map (ISDMAP)

CZAPB and DISPDUMP reference the ISDMAP to locate the ISD for each module named in a PCS statement.

Display List (DISPLIST)

The DISPLAY/DUMP component forms a display list for communication between DISPLAY/DUMP routines. The information used in forming DISPLIST is obtained from the phrase list whose entry point is passed to DISPDUMP as a parameter, and from the ISDMAP.

DISPLIST contains header information and an item (two items if a range is involved) for each entry in the phrase list. The header information identifies the action to be taken as DISPLAY, DUMP, or SET. It stores the phrase list entry location, the ISD map index qualification, and other housekeeping information. The body of the list identifies the items which are to be dumped, displayed, or set.

A full description of DISPLIST is given in Appendix F.

PHASE III MESSAGE FORMATS

Phase III issues confirmation and diagnostic messages in the following formats:

1. At symbol psw statement no. - a standard header, printed everytime a dynamic statement is processed (and, if conditional, is true).

   symbol

   the instruction location is expressed as an internal symbol with offset if the ISD for the module has been loaded. In assembly language programs, the closest internal symbol is used. In FORTRAN programs the nearest statement number is used. The statement number may be subscripted to indicate which un-

numbered statement following the numbered statement has control. If the ISD is not available, the instruction location is expressed as an external symbol (with offset). The external symbol used will be the control section containing the instruction location. If the instruction location is not in a control section, the location is expressed as a hexadecimal address.

psw

user's current PSW edited as:

```
PSW abc ffffffff
where
a = 2-bit ILC
b = 2-bit CC
c = 4-bit program mask
ffffffff = 32-bit address
```

statement no.

an integer, assigned to the PCS statement.

This message serves to correlate outputs to SYSOUT with the program flow; it always precedes such output.

2. STOP AT symbol psw - this message is printed when a STOP is processed in either a dynamic or immediate statement.

   symbol

   same as 1

   psw

   same as 1. The address portion contains the location of the next instruction to be executed.

3. RUNNING FROM symbol - this message is output when a GO or BRANCH command is processed.

   symbol

   same as 1

4. ILLEGAL ENTRY INTO PCS, instr, psw - this message is printed when a matching address is not found in the location table (LOCTAB).

   instr

   will be either
   EX R, D(X,B) if ILC = 2
   or SVC D if ILC = 1
   where R, D, B, and X are printed as decimal integers

   psw

   user's VPSW. The address contains the location immediately following the instruction that caused the error.

5. IMPROPER ORDERING OF DYNAMIC STATEMENT statement no. - this message is printed when a BRANCH or STOP phrase in a dynamic statement is processed and a subsequent dynamic statement for the same location is still outstanding.

   statement no.

   the PCS statement number of the outstanding statement.

6. Other diagnostics will be preceded by the standard header if the error occurs in a dynamic statement.

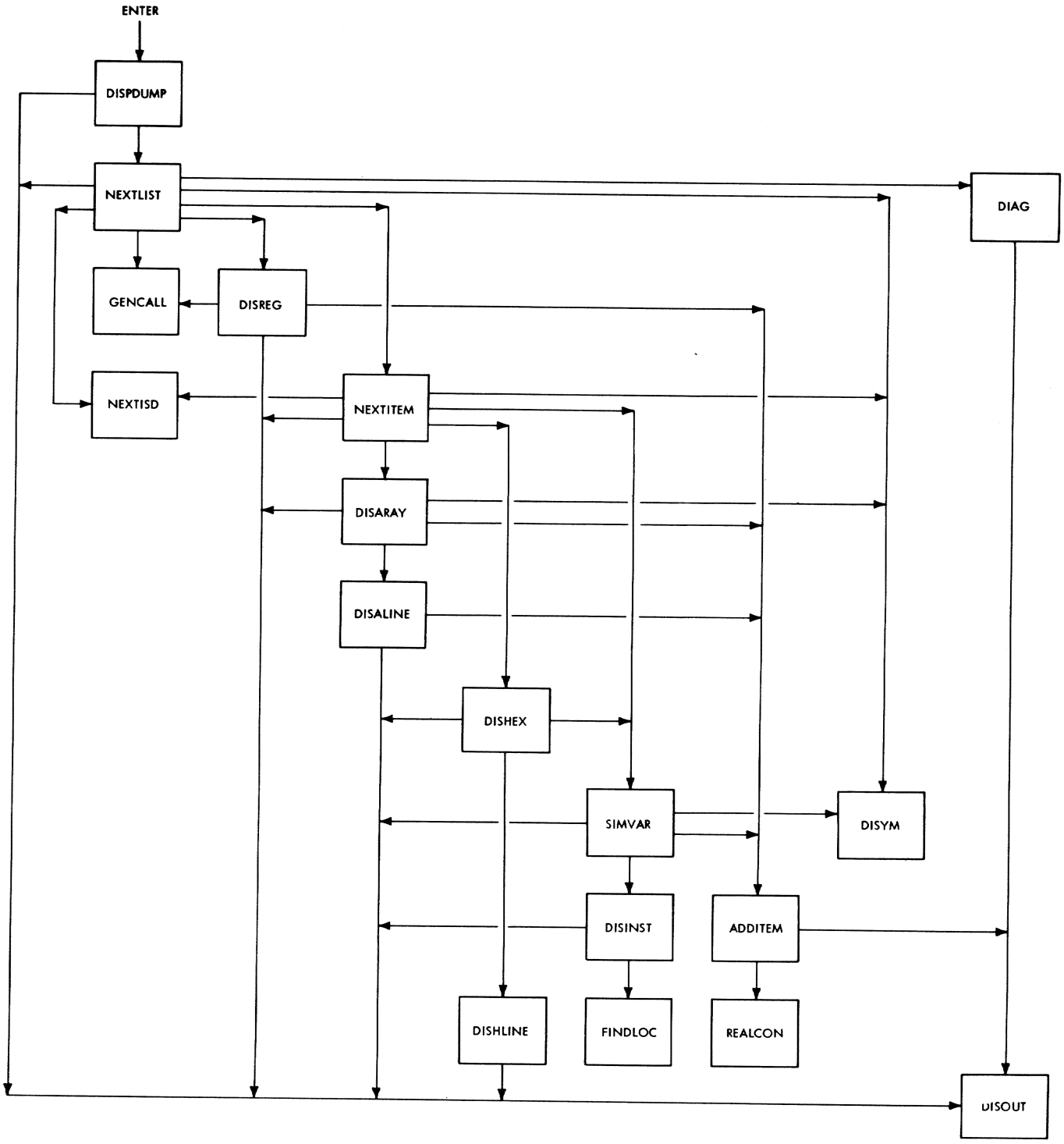


Figure 15. Display/Dump Nesting Chart

The descriptions presented in this section are arranged alphabetically by routine identification. They should be read and used along with the flowcharts in Section 4. A few flowcharts have been omitted, because these routines are adequately explained by their description.

Appendix B can be used to find the routine mnemonic corresponding to the routine identification since it is in subroutine ID order. However, Appendix A, which is in order by routine mnemonic, will provide a cross-reference to the routine identification.

CZAMA -- SET

This routine controls the processing of the SET phrase. (See Chart AA.)

Entry

CZAMA1
Entered via a type-I call with no parameters.

Routines Called

DATAFLD (CZAMI1)
Forms a data field item.

EXPSCAN (CZAMH1)
Expression scan.

DIAGNO (CZANW1)
Issued diagnostics.

GETPAGE (CZANZ1)
Allocates a page of virtual storage.

Exit

This subroutine returns control to the caller.

Operation

A phrase list header (PLHEAD) is created which identifies the phase as SET. Each SET phrase consists of a SET operand and an expression.

DATAFLD is called to form a field item for the SET operand whose length attribute is the length of the expression result. DATAFLD then creates a FLDITEM for the first term of the expression. EXPSCAN is called to form a Polish string. Next, the SET operand and the resulting expression are checked for length agreement. The phase list entry is now inserted in the phrase list. Formation of phrase list entries continues until the entire SET phrase has been processed.

The PLHEAD is then inserted, preceding the first phrase list entry, and a list terminator is placed immediately following the last entry.

CZAMB -- BRANCH

This routine controls the processing of the BRANCH phrase. (See Chart AB.)

Entry

CZAMB1
Entered via type-I linkage with no parameters.

Routines Called

DATAFLD (CZAMI1)
Forms a data field item.

DIAGNO (CZANW1)
Issues diagnostics.

GETPAGE (CZANZ1)
Allocates a page of virtual storage.

Exit

This routine returns control to the caller.

Operation

A phrase list header (PLHEAD) is formed, identifying the phrase as BRANCH. DATAFLD is called to form a data field item (FLDITEM) for the BRANCH operand. A BRANCH phrase list is then formed and linked to the immediate statement table entry (STATAB). If the statement is dynamic, the list is assigned storage space and is inserted in the phrase list page. If the BRANCH is immediate, and there is no active corresponding user program, a diagnostic is issued, and control is returned to the calling program.

Since the BRANCH phrase, when used, must be the last phrase in a PCS statement, a flag is set so that CA&E will delete any remaining characters in the source line.

CZAMC -- STOP & GO

This routine controls the processing of a STOP or a GO phrase. (See Chart AC.)

Entry

CZAMC1
Entered, for a STOP phrase, by type-I linkage with no parameters.

CZAMC2
Entered, for a GO phrase, by type-I linkage with no parameters.

Routines Called

DIAGNO (CZANW1)
Issues diagnostics.

Exit

This routine returns control to the caller.

Operation

A phrase list header (PLHEAD) is formed, which identifies the phrase as either STOP or GO. The PLHEAD is then linked to the immediate statement table entry (STATAB). If the statement is dynamic, the header is inserted in the phrase list page. If the STOP or GO phrase is immediate, and there is no corresponding active user program, a diagnostic is issued.

Since a STOP or GO phrase, when used, must be the last phrase in a PCS statement, a flag is set so that CA&E will delete any remaining characters in the source line.

CZAMD -- DISPLAY & DUMP

This routine controls the processing of a DISPLAY or a DUMP phrase. (See Chart AD.)

Entry

CZAMD1
Entered, for a DISPLAY phrase, by type-I linkage with no parameters.

CZAMD2
Entered, for a DUMP phrase, by type-I linkage with no parameters.

Routines Called

DATAFLD (CZAMI1)
Forms a data item.

EXPSCAN (CZAMH1)
Expression scan.

DIAGNO (CZANW1)
Issues diagnostics.

GETPAGE (CZANZ1)
Allocates a page of virtual storage.

Exit

This routine returns control to the caller.

Operation

A phrase list header (PLHEAD) is formed, identifying the phrase as either DISPLAY or DUMP. DATAFLD is called to form a data item for the first operand. If the operand is a constant, or the first term of an expression, EXPSCAN is called to form a Polish string. A phrase list entry is formed and inserted in the phrase list. Formation of phrase list entries continues until the entire operand has been processed. The PLHEAD is then inserted preceding the first phrase list entry, and a list terminator is placed immediately following the last phrase list entry.

CZAME -- IF

This routine controls the processing of an IF phrase. (See Chart AE.)

Entry

CZAME1
Entered by type-I linkage with no parameters.

Routines Called

DATAFLD (CZAMI1)
forms a data field item.

EXPSCAN (CZAMH1)
Expression scan.

DIAGNO (CZANW1)
Issues diagnostics.

GETPAGE (CZANZ1)
Allocates a page of virtual storage.

Exit

This routine returns control to the caller.

Operation

A phrase list header (PLHEAD) is formed identifying the phrase as an IF phrase. DATAFLD is called to form a data field item (FLDITEM) for the first term of the expression. EXPSCAN is called to form a Polish string for the expression. An IF phrase list entry is then created and linked to the immediate statement table entry (STATAB). If the statement is dynamic, the list is assigned storage by means of GETPAGE and inserted into the assigned phrase list page.

CZAMF -- AT

This subroutine controls the processing of an AT phrase. (See Chart AF.)

Entry

CZAMF1
Entered by type-I linkage with no parameters.

Routines Called

DATAFLD (CZAMI1)
Forms a data field item.

CODEGEN (CZANF1)
Generates code to evaluate offset.

GENCALL (CZAPN1)
Executes generated code.

GETPAGE (CZANZ1)
Allocates a page of virtual storage.

FREEMAIN
Releases allocated virtual storage.

DIAGNO (CZANW1)
Issues diagnostics.

MAPSEARCH (CZCCQ)
A dynamic loader subroutine which locates the CSD for the instruction.

CKCLS
Checks storage protection class.

Exit

This routine returns control to the caller.

Operation

A phrase list header (PLHEAD) is formed, identifying the phrase as an AT phrase. DATAFLD is then called to form a data item for each instruction location. If the instruction location is offset, CODEGEN is called to generate the code necessary to evaluate the offset, and GENCALL is called to execute the generated code. The instruction location address is then incremented by the computed offset.

An AT phrase list entry is formed for each instruction location and is inserted in the phrase list. Creation of phrase list entries continues until all operands of the AT phrase have been processed. The PLHEAD is then inserted, preceding the first phrase list entry, and a list teminator is placed immediately following the last list entry.

An entry for the statement is inserted into the dynamic statement table, (STATAB), and the dynamic statement flag is set.

CZAMG -- CALL

This routine controls the processing of the CALL phrase. (See Chart AG.)

Entry

CZAMG1
Entered for an explicit CALL, by type-I linkage with no parameters.

CZAMG2
Entered, for an implicit CALL, by type-I linkage with no parameters.

Routines Called

DATAFLD (CZAMI1)
Forms a data field item.

EXPSCAN (CZAMH1)
Expression scan.

DIAGNO (CZANW1)
Issues diagnostics.

GETPAGE (CZANZ1)
Allocate a page of virtual storage.

EXTERNAL (CZAM01)
Evaluates external symbol.

Exit

This routine returns to the caller

Operation

A phrase list header (PLHEAD) is formed, identifying the phrase as CALL. For an explicit CALL, DATAFLD is called to form a data field item for the entry point. For an implicit CALL, EXTERNAL is called to evaluate the module name in New Task Common. BUILTIN, a procedure key, is constructed containing, the entry point, the RCON for the entry point, and a parameter count of zero.

If a parameter list is specified in the phrase, DATAFLD is called to form a data field item (FLDITEM) for the first operand of each parameter. EXPSCAN is called to form a Polish string for each parameter, the parameter count is incremented and the address of the Polish string is inserted in the phrase list.

A CALL phrase list is formed. When all parameters have been processed, the phrase list is assigned storage in the phrase list page, if the statement is dynamic.

CZAMH -- EXPSCAN (EXPRESSION SCAN)

This routine scans and identifies the operators and operands of an expression, in order from left to right, and forms a Polish string. (See Chart AH.)

Entry

CZAMH1
Via standard type-I linkage.

Routines Called

GETPAGE (CZANZ1)
Allocates a page of virtual storage.

DATAFLD (CZAMI1)
Forms a data field item.

DIAGNO (CZANW1)
Issues diagnostics.

SCANFLD (CZAMQ1)
Scans a field to a delimiter.

PROMPT (CZANX1)
Issues a message and receives a response.

Exit

This routine returns control to the caller.

Operation

The first term of an expression will have been identified and defined by the caller. Subsequent operands are identified and defined through calls to DATAFLD, and are entered in a Polish string in the order in which they are encountered. Operators are identified as the delimiter of an operand, or by means of calls to SCANFLD.

Operators are placed in an operator stack with an assigned weight or value. Before an operator is entered into the operator stack, its assigned weight is compared to that of the previous operator. If the weight of the current operator is greater than, or equal to, the weight of the previous operator, the previous operator is removed from the stack and placed in the Polish string. This process is continued until the current operator can be placed in the operator stack (i.e., the current operator's assigned weight is less than that of the previous operator).

When a left parenthesis (is encountered in the expression, it is entered directly into the operator stack, and a parenthesis counter is incremented. When a right parenthesis) is encountered, operators are removed from the operator stack and placed in the Polish string until a left parenthesis operator is encountered. The left parenthesis operator is then removed from the operator stack, the parenthesis count is decremented, and the parenthesis operators are eliminated. When the end of the expression is reached, any operators remaining in the operator stack are placed in the Polish string.

Each entry in the resulting Polish string is identified as an operand or an operator. Operand entries further are identified as either constant or variable; registers are considered variable. PCS will provide storage for operands identified as constant. Each operand will also be identified as to data length and data type. The operand data

type is used to determine the data type of the entire expression.

Constant operands are variable operands are processed uniquely. The data type of a constant operand is always defined and automatically defines the data type of the expression in which it occurs. All constants within an expression must have the same data type, or the expression will be rejected.

The data types of all fully defined variable operands should agree with the data type of any constant operand occurring in that expression. If they do not agree, the data type of the constant operand(s) is assumed for the whole expression, and a diagnostic is issued.

If the expression contains variable operands only, the data types of all fully defined variables must agree in order to define the data type of the expression. In conversational mode, if the data types of variables do not agree, the user is prompted for data type definition information. The user is always prompted if the expression contains two or more variable operands, whose data types are undefined.

In non conversational mode, diagnostics cause the statement to be ignored.

When the logical operators (AND or OR), are encountered, the data type of the expression is reset.

If an operand is identified as a subscripted array, or as a symbol with offset, the subscript/offset Polish string is included in the Polish string for the expression being scanned, and a subscript/offset operator is entered in the operator stack.

Operator stack entries will contain an encoded value unique to the following operators:

Operator	Definition
(+)	Subscript/offset
+	Addition
*	Multiplication
/	Division
>	Greater than
>=	Greater than or equal to
=	Equal to
¬=	Not equal to
<	Less than
<=	Less than or equal to
¬>	Not greater than
¬<	Not less than
&	And
|	Or

Subtraction is accomplished through an addition operator and a unary arithmetic indicator, which is attached to the next operand in the expression. The NOT (¬) operator is a logical unary operator and is attached to the next operand. Unary operators are cleared after they have been attached to an operand. A plus or minus sign may be treated as a unary arithmetic operator in certain cases. For example: -A*B.

A plus unary operator is ignored. A minus unary operator is attached to the next operand. If the next operand is a parenthesized subexpression, the unary indicator is attached to the major operator of that subexpression. This is accomplished by attaching the unary indicator to the left partnthesis and entering the left parenthesis in the operator stack. When the corresponding right parenthesis is encountered, the unary indicators are attached to the last element in the Polish string.

The expression scan is terminated when an operand is not followed by one of the logical or arithmetic relational operators, and the parenthesis count is zero. Control is returned to the calling program.

The following error checks are made:

1. No two operators appear in sequence.

2. No two operands appear in sequence.

3. An expression may not be preceded by a multiplication or division operator (* or /).

4. Data types in an expression must agree. Undefined data types are assumed to agree.

5. Valid operands are expressed in terms of data location, either with or without offset or offset range.

6. For every left parenthesis, there is a corresponding right parenthesis. For every right parenthesis, there is a leading left parenthesis.

7. At least one operand appears in an expression.

CZAMI -- DATAFLD - FORM DATA FIELD DEFINITION

This routine controls the formation of a data field item (FLDITEM). (See Chart AJ.)

Entry

CZAMI1
Via standard type-I linkage.

Routines Called

CKCLS
Determines if a range of virtual storage locations are allocated to the user.

DATALOC (CZAML1)
Forms a data location item.

SUBPOL (CZAMJ1)
Evaluates subscripts and forms a Polish string.

DIAGNO (CZANW1)
Issues diagnostics.

Exit

This routine returns control to the caller.

Operation

A call is made to DATALOC to identify and define a data location. If subscript/offset notation is used, SUBPOL is called to form a Polish string for the subscript/offset expression. The data location item (LOCITEM) becomes the data field item (FLDITEM). If range notation is used, a second call to DATALOC is issued, to identify and define another data location. If subscript/offset notation is used on the second data location, SUBPOL is called upon to form a Polish string for the subscript/offset. The second data location is incorporated in the data field item. A branch is made, based on the type of LOCITEM last formed.

External and Hexadecimal Address: A check is made to ensure that the data location virtual storage address is larger than, or equal to, that of the data field, and that all storage between those addressed is assigned.

Internal, Statement Number, and Sub-Scripted Array: A check is made to ensure that the VMA of the data location and the data field are in the same control section, and that the data location's VMA is larger than, or equal to, that of the data field.

Hexadecimal, Integer, Character, Address, or Floating Point Constants: The FLDITEM is identified as a constant, and a check is made to ensure that range notation was not used.

Register: A register range item is formed and a check that integers less than 16 are used to specify general registers, and that even integers less than 8 are used to specify floating point registers, is made.

Error: The data field item is identified as an error.

Null and Percent Count: A check is made to ensure that range notation was not used.

Command Variable and Undefined Variable: No processing is performed.

CZAMJ -- SUBPOL - SUBSCRIPT TO POLISH

This routine identifies the operands and operators of a subscripted or offset expression, and forms a Polish string. (See Chart AK.)

Entry

CZAMJ1
Via standard type-I linkage.

Routines Called

GETPAGE (CZANZ1)
Allocates a page of virtual storage.

DATALOC (CZAML1)
Forms a data location item.

SCANFLD (CZAMQ1)
Scans a field to a delimiter.

DIAGNO (CZANW1)
Issues diagnostics.

VALMOD (CZAOA1)
Evaluates module name.

CKCLS

Checks storage protection class.

Exit

This subroutine returns control to the caller.

Operation

Initially, a left parenthesis pseudo-operator is entered into the operator stack. If a subscripted array is being processed, a dimension constant is entered into the Polish string, and a dimension operator is entered into the operator stack.

Operands are identified and defined through calls to DATALOC. If the operand is identified as a hexadecimal address, and range notation is used, a second call is made to DATALOC to identify and define the upper limit of the range.

CKCLS is called to ensure that the storage locations of all operands are assigned. The operand item is then entered in the Polish string.

If the data type of an operand has been defined, and it is not defined as either integer or hexadecimal, a diagnostic is issued.

Operators are entered into the operator stack with an assigned weight or value. Before an operator is entered into the operator stack, its assigned weight is compared to that of the previous operator. If the weight of the current operator is greater than, or equal to, the weight of the previous operator, the previous operator is removed from the stack and put into the Polish string. This process is continued until the current operator can be entered into the operator stack. That is, the current operator's assigned weight is less than that of the previous operator.

Operator entries will contain an encoded value unique to the following operators:

+	Addition
*	Multiplication
/	Division

Subtraction is accomplished through an addition operator and a unary arithmetic operator attached to the next operand.

When a left parenthesis (is encountered, it is entered directly into the operator stack, and a parenthesis counter is incremented.

When a right parenthesis) is encountered, operators are removed from the operator stack and entered into the Polish string, until a left parenthesis operator is encountered. The left parenthesis operator is then removed from the operator stack, the parenthesis count is decremented, and both parenthesis operators are discarded.

The comma is a special operator. When a comma is encountered, the operator stack is emptied and its contents are entered into the Polish string. When this happens, the parenthesis count must be zero, or a diagnostic is issued. If a subscripted array is being processed, the next dimension constant is entered into the Polish string and a dimension operator is entered into the operator stack. A dimension operator has the highest priority of any operator in the operator stack.

If a second comma in an offset is detected, the routine searches for a type field, and if it finds a valid one, sets the appropriate code in LOCOUTYP. The source list is then adjusted to make it appear as though no type field existed. Processing continues as for an offset without type indicator.

When a subsequent comma or an end-of-subscript indicator is encountered, the dimension operator is removed from the operator stack and entered into the Polish string as a plus operator. If a symbol with offset is being processed, the comma indicates offset range notation and DATALOC is called to identify and define the offset range. The offset range must be expressed as either an integer or a hexadecimal constant and must be delimited by a right parenthesis (end-of-offset indicator). The offset range is incorporated in the item for the symbol with offset.

An end-of-subscript/offset condition occurs when a right parenthesis is encountered at parenthesis level zero; (i.e., the initial left parenthesis pseudo-operator has been removed from the operator stack). The subscript/offset Polish string is then terminated.

Although this subroutine is not recursive, nested subscripts and/or offsets are permitted. Operands in a subscript/offset expression are identified by DATALOC. Upon entry, the DATALOC item for the subscripted array/symbol with offset is entered in a nested stack. DATALOC items are formed for the operands in the subscript/offset expression. If an operand in a subscript/offset expression is subscripted/offset, the DATALOC item for the operand is entered in the nested stack. When an end-of-subscript/offset condition is detected, a subscript/offset operator is entered in the operator stack. The last entry made in the nested stack is removed and processed as a simple operand. When all entries have been removed from the stack, the subscript/offset Polish string is terminated.

The following error checks are made:

1. No two operators appear in sequence.
2. No two operands appear in sequence.
3. Data types must be integer, hexadecimal, or undefined.
4. Data lengths must be 1, 2 or 4 bytes.
5. Commas within the subscript must not be enclosed in parentheses.
6. The initial character of a subscript, or the character immediately following a comma, must not be a * multiplication or a / division operator.

CZAML -- DATALOC - FORM DATA LOCATION

This routine analyzes the syntax of an expression, identifies the data location, and controls the evaluation and formation of data location and source list items. (See Chart AL.)

Entry: CZAML1--Via standard type-I linkage.

Routines Called: SCANFLD (CZAMQ1)--Scans a field to a delimiter.

VALMOD (CZAOA1)--Evaluates module name.

VALSYM (CZAOB1)--Evaluates internal symbol from ISD.

EXTERNAL (CZAMO1)--Evaluates external symbol.

DIAGNO (CZANW1)--Issues diagnostics.

GETCHAR (CZAMQ2)--Gets next character.

Exit

This subrouting returns control to the calling program.

Operation

The data location item LOCITEM and the four identified source list items PQNITEM, SQNITEM, SYMITEM, and SUBITEM are initialized, and SCANFLD is called to form a source list item (SLITEM). An SLITEM consists of a character string (identified as null, numeric or alphabetic) and a delimiter. Alphameric strings are processed as alphabetic strings.

The identification and delimitation of the source list item are inspected to determine the format of the data location. Depending upon the format used, several source list items may be required in order to completely identify the data location. During the process of identification, the source list item formed is moved to one of the four identified source list items and if necessary, they may be adjusted to accommodate the new source list item. The valid contents of the identified source list items, are shown in Figure 16.

ABC, MNO, and XYZ represent, respectively, the first, second and third alphabetic or alphameric character strings encountered. Similarly, 123, 456, and 789 represent numeric strings. Any combination of source list items other than those specified in Table 1, is an error and will result in a diagnostic.

When a data location has been identified, its context is checked. If the context is not valid an error diagnostic is issued. If the data location is valid in the current context, control is passed to the appropriate conversion and/or evaluation routine.

Two indicators IMPLICIT and EXPLICIT are set in SYMIND to aid in the evaluation of symbols. The IMPLICIT indicator is set for an implicitly qualified symbol (i.e., an array, a statement number, or the special %CSECT symbol) while the EXPLICIT indicator is set for an explicitly qualified symbol.

Array, Statement Number, and Statement Number with Offset: VALMOD is called if the internal symbol is explicitly qualified. VALSYM is called to evaluate the internal symbol.

Table 1. Syntax Analysis Table

Data Location	PQNITEM		SQNITEM		SYMITEM		SUBITEM		SLITEM	
Array	ABC ABC	. .	MNO	.	ABC MNO XYZ	(((				
Internal with offset	ABC ABC	. .	MNO	.	ABC MNO XYZ	. . .				(((
Internal	ABC ABC	. .	MNO	.	ABC MNO XYZ					
Statement number with offset	ABC ABC ABC ABC	. . . .	MNO MNO	. .	123 123 123 123 123 123	. (. (. (	456 456 456	)))	. . .	((((((
Statement number	ABC ABC ABC ABC	. . . .	MNO MNO	. .	123 123 123 123 123 123	(((	456 456 456	)))	-	
External					ABC					
External with offset					ABC	. .				((
General register					123	R				
Single precision reg					123	E				
Double precision reg					123	D				
Hexadecimal address					L	'				
Hexadecimal constant					X	'				
Character constant						'				
Address constant					A	'				
Integer constant					123					
Percent count					%					
Null										
Single precision constant		 + - + - + - + -	456 456 456 456 456 456 789 789 789 456 456 456		123 123 123 123 123 123 123 123 123 123 123 123 123 123	. . . E E E	456 456 456 123 123 123 456 456 456 123 123 123 456 123	 E E E E E E E E E		
Double precision constant		 + - + - + - + -	456 456 456 456 456 456 789 789 789 789 789 789		123 123 123 123 123 123 123 123 123 123 123 123 123 123	D D D	456 456 456 456 456 456 456 456	 D D D D D D D D D D D D		

Internal and Internal with Offset: VALMOD is called if the symbol is explicitly qualified. VALSYM is called to evaluate the internal symbol. If the evaluation is unsuccessful and the symbol is not implicitly or explicitly qualified, the symbol is processed as an external or external with offset.

External, External with Offset, and Offset: EXTERNAL is called to evaluate the symbol. An offset, [indicated by .() not preceded by a name is assumed to be an offset from zero if no qualification is in effect, and is assumed to be an offset from the qualified name if a qualification is in effect. EXTERNAL is called to handle either case as if it were external with offset.

Address Constant: If the address constant indicator is cleared, the valid operand indicators (designated by the program symbol EVALUATE) are set to allow arrays, statement numbers, internal symbols, and external symbols to be identified in the current context. The address constant indicator is set and processing continues with the formation of the source list item. If the address constant indicator is set, the data location is evaluated as an internal symbol.

Binary Constant: Binary is handled internally as for a hexadecimal constant. The binary flag is set in LOCOUTYP, and the binary input stream in the source list is converted to hex.

Hexadecimal Address and Hexadecimal Constant: Once it is determined that hexadecimal information is being processed, subsequent characters are scanned until the terminal quote is recognized. The characters are then converted to hexadecimal and the data location item is formed. Hexadecimal address with offset will be treated the same as external with offset, the hexadecimal address itself being substituted for the virtual memory address that would be obtained from EXTERNAL. This is also true for ranges which combine hexadecimal address with or without offset and external or internal with or without offset.

Character Constant: Once it is determined that character information is being processed, subsequent characters are scanned until the terminal quote is recognized and the data location item is formed.

Integer Constant: The SYMITEM character string is converted to integer and the data location item is formed.

General Register, Single Precision Register, and Double Precision Register: The SYMITEM character string is converted to integer and the data location item is formed.

Single Precision Constant and Double Precision Constant: The SYMITEM and SUBITEM character strings are first converted to integer and then to an unnormalized floating point number. The SQNITEM is converted to integer, the exponent is adjusted, and the floating point number is raised to the exponential power. The data location item is then formed.

Percent Count: The data location item is formed.

Null: A null data location item is formed.

CZAMO -- EXTERNAL - FORM EXTERNAL SYMBOL

This routine completes the formation of a data location item (LOCITEM) for an external symbol, a defined command variable, or an undefined command variable. (See Chart AM.)

Entry: CZAMO1--Via standard type-I linkage

Routine Called: Dynamic loader is called by the following hand-coded expansion of the LOAD macro instruction.

In PSECT

```
LOADM   DLINK
        DC      X'8001'
LNAME   DC      CL8' '
V       DC      A(0)
R       DC      A(0)
```

In CSECT

```
        EX      0,LOADM
```

HASHSEARCH (CZCDL2)--A Dynamic Loader subroutine which locates the CSD for the external symbol with the calling sequence:

CALL CZCDL2,(LIST)

where LIST consists of five parameters:

1 - Pointer to the hash table.
2 - zero
3 - Pointer to the symbol name.
4 - Contains the module sequence number.
5 - Contains the VMA of the symbol definition (on return).

Dictionary handler (CZASD3)--Locates a defined command variable with the calling sequence:

CALL CZASD3,LIST

where LIST consists of five parameters:

1 - Pointer to the location of the combined dictionary address.
2 - Pointer to the command variable name.
3 - Pointer to the entry mask.
4 - Pointer to the hash value of the name.
5 - Pointer to the address of the entry (on return).

Exit: This routine returns to the calling program.

Operation: HASHSEARCH is called to locate the symbol definition. If the symbol is a system symbol, the first word of the hash search parameter list points to the system hash table; otherwise, the first word points to the user hash table. If the symbol definition is found, the data location item for the external symbol is completed. If necessary, the virtual storage address of the PMD is inserted in the immediate table of modules referenced by PCS and the routine exits.

If the symbol definition is not found, the dynamic loader is called to load the module. If the module was loaded correctly, HASHSEARCH is called to locate the definition. If the definition is found, the data location item is completed as described above, and the routine exits. If the definition is not found after the second call to HASHSEARCH, a diagnostic is issued and the routine exits.

If the module was not correctly loaded by the dynamic loader, the current context is checked. If an undefined command variable is valid in the current context, a data location item for an undefined command variable is completed and the routine exits. If a defined command variable is valid in the current context, the Dictionary Handler is called to locate the combined dictionary entry for the command variable. If the entry is found, the data location item for the defined command variable is completed and the routine exits. If, however, the command variable entry is not found, or a command variable is not valid in the current context, a diagnostic is issued.

CZAMQ -- SCANFLD & GETCHAR

This routine scans the input source statement to a field delimiter, and forms a source list item (SLITEM) for the field scanned. (See Chart AN.)

Entry: CZAMQ1 (SCANFLD)--Via standard type-I linkage.

CZAMQ2 (GETCHAR)--To get the next character to be scanned, from the source list, and return with the character in register 1.

Routines Called: DICTIONARY HANDLER (CZASD3)--Locates a synonym with the calling sequence: CALL CZASD3,LIST

where LIST consists of five parameters:

1 - Pointer to the location of the combined dictionary address.
2 - Pointer to the synonym.
3 - Pointer to the entry mask.
4 - Pointer to the hash value of the synonym.
5 - Pointer to the address of the entry (on return).

SOURCE LIST HANDLER (CZASC3, CZASC4, CZASC5)--Inserts a synonym into the source list with the calling sequence:

CALL CZASC5,LIST

where LIST consists of one parameter:

1 - Pointer to the address of the synonym entry in the combined dictionary.

The Source List Handler is called in the GETCHAR subroutine to process source list markers with the calling sequence:

CALL CZASC3 and CALL CZASC4.

Exit: This routine returns to the calling program.

Operation: If a source list item (SLITEM) is outstanding, the position pointer in the PCS diagnostic buffer is updated and processing continues with checks for synonym substitution as discussed below. (An outstanding source list item may occur as a result of a relational operator in an expression, or as a result of the syntax analysis of a floating point register or constant.)

If a source list item is not outstanding, the source list item is initialized. The source list item contains the character string identification, the address of the first character in the string, the length of the character string, and the delimiter. The address of the next available byte in the PCS diagnostic buffer becomes the starting address of the character string. The length and identification characteristics of the string are initialized as zero and null respectively.

Characters are extracted from the source list via GETCHAR; they are classified; and control is passed to the appropriate subroutine based on the character class. These subroutines determine whether the character in question is a continuation character or a delimiter. If it is a continuation character, it is identified as either numeric or alphameric, the length of the character string is incremented, and the character is concatenated with the previous characters. If the character is a delimiter, it is stored in the source list item and is also concatenated with the previous characters. Delimiting characters do not participate in the identification of the character string, nor are their lengths included in the length of the string.

When a delimiter is encountered, checks are made to see if synonym substitution is possible. If the "synonym possible" indicator is set, alphameric strings of eight characters or less are hashed, and the Dictionary Handler is called to search for a synonym. If a synonym is found, the delimiter is restored to the source list. (The delimiter will be extracted from the source list after the synonym string has been scanned.) At this point, the Source List Handler is called to insert the synonym in the source list, and the source list scan is re-initiated.

The following examples illustrate the treatment of synonym substitution by PCS. It is assumed that the reader is familiar with the structure of PCS, since these examples deal with program logic outside the scope of this subroutine.

Example 1: The source list contains the string: ALPHA, BETA where both ALPHA and BETA are candidates for synonym substitution. Suppose ALPHA has a synonym PGMA.ALPHA. After the substitution, the source list contains the string: PGMA.ALPHA,BETA.

In the above string, only PGMA and BETA are candidates for further substitution.

Example 2: The source list contains the string: RESULT,BETA and, as in example 1 above, both RESULT and BETA are candidates for synonym substitution. Suppose RESULT has a synonym of A+B. After the substitution, the source list would contain the string: A+B,BETA.

In the above string, A, B, and BETA are all candidates for further synonym substitution. Notice that the delimiter of RESULT becomes the delimiter of B.

Example 3: The source list contains the string: 100, 10R, 2E+1

In this example, there are no candidates for synonym substitution since all the initial strings are numeric.

Example 4: The source list contains the string: A'ALPHA' where both A and ALPHA are candidates for synonym substitution. If ALPHA has the synonym PGMA.ALPHA and there is no synonym for A, the resultant source list string would be: A'PGMA.ALPHA'

PGMA is a candidate for further synonym substitution.

Example 5: The source list contains the string: L'FACE,X'FACE','FACE'

In this example, the strings L and X are candidates for substitution, while the three character strings, FACE, are not.

CHARACTER CLASSES: The following character classes have been established.

Class 0 Any character not specified in the remaining classes. These are continuation characters, and the source list item identification indicates the presence of an alphameric character.

Class 1 [0 thru 9] These characters are continuation characters. The source list item is identified as numeric.

Class 2 Space This is a continuation character. If it is encountered in the initial position of a source list string, it is discarded. If a continuation character has been processed prior to the occurrence of a space, the space is concatenated with the previous characters, the character string length is incremented, and the position in the PCS diagnostic buffer is temporarily updated. An indicator is set to identify a trailing blank. Then, if a delimiter is the next non-space character encountered, all trailing blanks are discarded. If the next non-space character is a continuation character, the character string length and PCS diagnostic buffer position are updated to include the space. If a string contains an embedded blank, an indicator is set, and the source list item identification indicates the presence of an alphameric character.

Class 3 DER These characters may be delimiters or continuation characters. If a numeric string is being scanned, these characters are considered to be delimiters. They are also treated as delimiters if a floating point constant is being scanned. In all other cases, they are processed as continuation characters.

Class 4 End-of-block and semicolon (;) These characters are delimiters.

Class 5 [+ - * / & | < > = ¬ ( ) . ' : ,] These characters are delimiters.

Class 6 Horizontal tab character.

GETCHAR

This routine extracts the next character from the source list via the GNC macro. If the extracted character is an end-of-block, and E marker has been encountered. If the E marker points to a procedure marker (P marker), the Source List Handler is called to process the marker. Upon return, the next character is extracted from the source list.

In all other cases, the character extracted from the source list is returned to the calling program in general register 1. The following error checks are made:

1. Alphabetic/alphameric strings are eight characters or less in length.
2. Alphabetic/alphameric strings do not contain embedded blanks.

CZAMR -- QUALIFY

This routine controls the processing of the QUALIFY phrase. (See Chart AO.)

Entry: CZAMR1--Entered by type-I linkage with no parameters.

Routine Called: SCANFLD (CZAMQ1)--Scans a field to a delimiter.

VALMOD (CZAOA1)--Evaluates module names.

DIAGNO (CZANW1)--Issues diagnostics.

Exit

This routine returns to the caller.

Operation

SCANFLD is called to form a source list item (SLITEM) for the primary qualifying name. If the delimiter is a period, SCANFLD is called again to form a source list item for the secondary qualifying name.

The ISDMAP entry for the module name is located by VALMOD. The entry number and location are then saved for subsequent use by VALSYM in evaluating implicitly qualified internal symbols, statement numbers, or offsets.

CZAMS -- REMOVE

This routine controls the processing of the REMOVE phrase. (See Chart AP.)

Entry: CZAMS1--Via type-I linkage with no parameters.

Routines Called

DATALOC (CZAML1)
Forms a data location item.

DIAGNO (CZANW1)
Issues diagnostics.

Exit

This routine returns to the caller.

Operation

DATALOC is called to form a data location item (LOCITEM) for each statement number in the list. The statement number is used as an index to the statement table (STATAB) for the purpose of locating the appropriate entry. Each entry in the AT list for the statement, is unlinked from the chain of statement table entries for the particular location. When only one statement table entry is active at a location, the location table entry (LOCTAB) is nullified and the user's instruction is replaced. If the parameter "ALL" is entered, this routine acts as though a parameter list "1,2,...n" (where n is the current statement count) had been entered.

CZAMT -- UNLOAD

This routine adjusts PCS tables and cleans up user's program when a module referenced in prior PCS statements is being unlinked by the dynamic loader. (See Chart AQ.)

Entry

Via a standard type-I linkage. Register 1 contains the address of the PMD preface for the module being unloaded.

Routines Called

FREEMAIN Macro
Releases storage no longer needed

PRMPT Macro
Issues messages to the user

MAPSEARCH (CZCCQ)

Exit

To the calling program (Dynamic Loader).

Operation

All the saved instructions in the location table (LOCTAB) are restored in the user's program, thus destroying the PCSVCs. Any ISDs that have been fetched are released. Storage allocated for the location table, statement table, ISD map, phrase list, Polish string, or generated code is released. If any dynamic statements were entered, a call to GATE notifies the user that his program has been restored.

CZANA -- PHASE2 - PHASE II PCS INPUT CONTROL

Phase II control is the control routine for the final processing of the statement phrases. (See Chart AR.)

Entry

CZANA1
Via standard type-I linkage.

Routines Called

CODEGEN (CZANF1)
Translates a Polish string to generated code.

VISAM OPEN
Opens PCSOUT data set on first DUMP command.

FINDLOC (CZAPC1)
Finds a LOCTAB entry for the instruction location.

FREEMAIN Macro
Releases storage no longer needed.

GATWR
Writes the statement number on SYSOUT.

PROMPT (CZANX1)
Issues prompting messages.

PCSPUT (CZAPB2)
Evaluates conditions and perform actions.

DIAGNO (CZANW1)
Issues diagnostic.

DICTIONARY HANDLER (CZASD3, CZASD5, CZASD6)
Locates a command variable with the calling sequence: CALL CZASD3,LIST

where LIST consists of five parameters:

1 - Pointer to address of combined dictionary address.
2 - Pointer to command variable name.
3 - Pointer to entry mask.
4 - Pointer to the hash value of the name.
5 - Pointer to the VMA of the entry (on return).

The Dictionary Handler also deletes a command variable entry with the calling sequence: CALL CZASD6

The Dictionary Handler also inserts a command variable entry in the dictionary with the calling sequence: CALL CZASD5,LIST

where LIST is a single parameter:

1 - Pointer to the VMA of the command variable entry.

Exit

This routine returns control to the calling program. If an immediate CALL phrase is being processed, Phase II Control passes control to the User Control routine (CZAMZ1) via a special transparent linkage so that when the User Control Routine issues a return, control returns to the program which called Phase II Control.

Operation

If any diagnostics were issued in Phase I, the diagnostic level is checked. If a fatal diagnostic was issued, DIAGNO is called to issue a diagnostic. CA&E is then instructed to delete the remainder of the line. Otherwise, control is passed to PROMPT to write the diagnostic and solicit user approval.

The statement table entry (STATAB) is located and initialized. The phrase list is located and each entry in the phrase list is inspected to see if code should be generated for the entry. CODEGEN is called to generate the code required. If a command variable is being defined or redefined via a SET phrase, the Dictionary Handler is called to locate the command variable entry in the combined dictionary. If an entry is not found, the Dictionary Handler is called to insert the entry. The dictionary handler is then called to locate the inserted entry.

If an entry for the command variable is in the combined dictionary, the data length of the entry in the dictionary is compared to the data length of the command variable being redefined. If the data lengths do not agree, the command variable entry in the dictionary is deleted, and the new command variable entry is inserted in the dictionary.

If the statement is immediate, PCSPUT is called to perform the actions specified by the statement. The storage assigned for the phrase list, generated code, and Polish string is then released.

If the statement is dynamic, FINDLOC is called to locate a LOCTAB entry for each AT list entry. If the LOCTAB entry is in use, the chain of STATAB entries for the location is followed until the last entry is found and the current STATAB entry is linked to the last entry. If the LOCTAB entry is not in use, an entry is created for the location and a PCSVC is inserted in the user's program. Storage for the STATAB entry, the phrase lists, and generated code is permanently assigned. Storage for the Polish string is released. The PCS statement number assigned to the statement is written out.

CZANF -- CODEGEN - CODE GENERATOR

This routine is the control routine for the specific operator code generation routines. (See Chart AS.)

Entry

CZANF1
Via standard type-I linkage.

Routines Called

GETPAGE (CZANZ1)
To allocate a page of virtual storage for generated code.

OPGEN (CZANI1)
To generate code to combine two operands.

SUBGEN (CZANG1)
To generate code for dimension computation.

GETBASE (CZANV1)
To convert a VMA to base displacement address.

LOADOP (CZANT1)
To generate code to load an operand.

COMCON (CZANH1)
To combine two constant operands.

SOURCE LIST HANDLER (CZASC2)
Expands the source list with the calling sequence: CALL CZASC2, LIST where list is a single parameter:

1 - Pointer to the number of pages needed to expand the source list.

Exit

This routine returns control to the calling program.

Operation

CODEGEN is the control routine for all object code generation. The Polish string is processed and, based on the types of entries, the appropriate routine is called to combine two operands.

Entries in the Polish string are inspected. The operands are placed in a pushdown list. As each operand is encountered in the Polish string, a pointer to the entry is placed in a pushdown list. When an operator is encountered, the last two operands are removed from the pushdown list. Control is given to the appropriate routine to generate the code for the operation. Selection of this routine is based on whether the operator is part of a dimension string. If it is, SUBGEN is called. If both operands are constants, COMCON is called to combine them via the operator into one Polish string entry. Otherwise, OPGEN is called to generate code to perform the operation.

Upon return from the specific code generation routine, the generated code result is placed in the pushdown list, and the next entry in the Polish string is inspected.

Processing of the Polish string continues until a trailer entry is encountered. CODEGEN completes its processing by providing the object code to return the final result of the computation.

In the sample expression A+B*(C-D) the final Polish string has the form:

```
HEADER
OPERAND    A
OPERAND    B
OPERAND    C
OPERAND   -D
OPERATOR   +
OPERATOR   *
OPERATOR   +
TRAILER
```

The processing of the Polish string would be accomplished as follows:

Step 1: The header is recognized as a control entry. Since the information in the header pertains to all the entries following, it is stored for later references.

Step 2: A is recognized as an operand. Its location is entered as the first entry in the pushdown list.

Step 3: Operands B, C, and -D are entered in the pushdown list as the second through fourth entries:

OPSTACK	Polish String
ENTRY 1 ⟶	OPERAND A
ENTRY 2 ⟶	OPERAND B
ENTRY 3 ⟶	OPERAND C
ENTRY 4 ⟶	OPERAND -D

Step 4: The + operator indicates that a combination must be performed. Entries 3 and 4 are removed from the pushdown list. Since C and -D are variables, OPGEN is called to generate code to perform the combination.

Step 5: OPGEN calls subroutines GETBASE, to make base register assignments, and LOADOP, to load the first operand (C) into a register. Since the second operand is negated, a subtract command will be generated. The resulting entry (C-D) is made in the Polish string in place of the original operand C entry. A pointer to the entry replaces entries 3 and 4 in the pushdown list:

OPSTACK	Polish String
ENTRY 1 ⟶	OPERAND A
ENTRY 2 ⟶	OPERAND B
ENTRY 3 ⟶	OPERAND C-D

Step 6: The * operator is encountered next. Again OPGEN is called to generate code to perform the multiplication between the last two entries in the pushdown list. The resultant operand replaces operand B in the Polish string, and entries 2 and 3 in the pushdown list:

OPSTACK	Polish String
ENTRY 1 ⟶	OPERAND A
ENTRY 2 ⟶	OPERAND B*(C-D)

Step 7: The final + operator causes the combination of the last two operands into the final result:

OPSTACK	Polish String
ENTRY 1 ⟶	OPERAND A+B*(C-D)

Step 8: The trailer entry signifies the end of the Polish string. CODEGEN determines if the result will be returned from the object code in the proper register. If necessary, CODEGEN generates the code to transfer the result followed by the return branch coding. For the subscript an array in the DISPLAY A (I, J) expression, the Polish string had the form:

```
FIRST DIMENSION OPERAND
OPERAND I
DIMENSION OPERATOR (*)
SECOND DIMENSION OPERAND
OPERAND J
DIMENSION OPERATOR (*)
PLUS OPERATOR
TRAILER
```

The processing of the Polish string would be accomplished as follows:

Step 1: Entries are made in the pushdown list for the first and second operands.

Step 2: The * operator encountered next is recognized as a dimension operator. SUBGEN is called to generate the code to perform the multiplication. In addition, code is generated to provide a dimension check of the result. A pushdown list entry of the result replaces the original operand.

Step 3: The second and third entries are made in the pushdown list for the next dimension operand and J.

Step 4: The * operator causes the computation of J* dimension operand as in step 2. The pushdown list is now:

OPSTACK	Polish String
ENTRY 1 ⟶	I*dimension#1
ENTRY 2 ⟶	J*dimension#2

Step 5: The + operator, not flagged as a dimension operator, causes OPGEN to be called, OPGEN generates the code to perform the addition between the two operands and form the result operand.

Step 6: The trailer entry causes CODEGEN to process the final operand and generate the return object code.

When the pushdown list contains a single operand, code is generated to return the resultant operand to the generated code caller.

If a call phrase is in process and the resultant operand is a constant, the constant is inserted in the last source list page. The constant is aligned on the appropriate boundary consistent with the data type of the constant. It is preceded with its length byte and followed by an E marker. After the constant is inserted in the source list, it is treated as a variable. The Source List Handler may be called to expand the source list in order to insert the constant.

There are three ways in which the result of generated code can be returned:

- The result can be loaded in general register 1.
- The result can be loaded in floating point register 0.
- A pointer to the result can be loaded in general register 1.

If code has been generated to load the result, if the result is to be negated either arithmetically or logically, or if the result cannot be returned in storage (i.e., the result of generated code and the recipient storage area are not the same length), the result is returned in a register.

If a base register is not assigned to an operand returned in storage, GETBASE is called to assign a base register for the result. If the base register assigned is not register 1, the following instruction is generated:

```
LA  1,D2(X2,B2)
```

To load the return code into general register 15 and exit, the following instructions are generated:

```
LA  15,8(0,0)
BR  14
```

This return code indicates that the result is returned in storage.

If the result is to be returned in a register, LOADOP is called to generate the code to load operands that are not in registers already. If the operand is logically negated, the following instructions are generated:

```
LA  10, 1
XR  10, R2
LR  1, 10
```

If the operand is arithmetically negated one of the following instructions is generated, depending upon the data type of the expression.

```
    LCR  1, R2
or LCDR 0, R2
```

If the result is not negated and is not in the correct register, one of the following instructions is generated, depending upon the data type.

```
    LR   1, R2
or LDR   0, R2
```

The following instructions are generated for results returned in general register 1:

```
LA   15, 0(0,0)
BR   14
```

The following instructions are generated for results returned in floating point register 0:

```
LA   15, 4(0,0)
BR   14
```

CZANG -- SUBGEN - SUBSCRIPT GENERATOR

This routine assists in the code generation for subscripting. (See Chart AT.)

Entry

CZANG1
Via standard type-I linkage

Routines Called

LOADOP (CZANT1)
Loads an operand.

DIAGNO (CZANW1)
Issues diagnostics.

GETBASE (CZANV1)
Assigns a base register to the ISD entry.

Exit

This routine returns control to the calling routine.

Operation

If the operator is multiply, the operands are the dimension operand and the subscript. When the array is a FORTRAN dummy argument, the dimensions are dummy arguments. The dimensions are in true form and not factors; therefore different processing is required.

Non-dummy Arguments: If the subscript is a constant, no code is generated. The dimension check is performed at this time. If the subscript exceeds the dimension, a diagnostic is formed.

If the subscript is not a constant, code is generated to compute the dimension. LOADOP is called to load the subscript operand. GETBASE is called to assign a base register to the ISD entry for the array. The instructions required to perform the multiply and the dimension check are then generated.

1.	L		(subscript generated by LOADOP0)
2.	BCTR	R1,0	(decrement subscript register)
3.	M	R1,D2(0,B2)	(multiply subscript register by dimension constant)
4.	CL	R1,D2+4(0,B2)	(compare subscript factor to next dimension factor)
5.	BC	4,D2(0,15)	(branch if result is in range) Note: The D2 field is set to branch to the instruction following instruction (7).
6.	LR	15,R2	(load ISD pointer into return code register) Note: The R2 field is the same register that is in the B2 field of instructions (3) and (4).
7.	BR	14	(return to caller)

Dummy Arguments

LOADOP is called to load the subscript. For the first dimension, a dimension factor operand is created. The dimension factor is entered in a stack.

The current dimension factor is located in the stack. If the dimension factor is not in a register, LOADOP is called to load the factor. The following commands are generated to compute the dimension and perform the dimension check.

1. L (subscript generated by LOADOP).

2. L (current dimension factor generated by LOADOP).

   Note: This command is always generated for the first dimension. It is generated for subsequent dimensions if they are stored subexpressions. (See GETREG.)

3. BCTR R1,0 (decrement subscript register).

4. MR R1,R2 (multiply subscript register by dimension factor register).

5. M R1,D2(0,B2) (multiply dimension factor register by next dimension).

   Note: GETBASE is called to assign a base register for the next dimension prior to the generation of this instruction. If the ISD indicates that the next dimension is a halfword, an MH instruction is generated.

6. CLR R1,R2 (compare subscript register with current dimension factor register).

7. BC 4,D2(15,0) (branch if result is in range)

   Note: The D2 field is set to branch to the instruction following instruction (9).

8. LA 15,16 (load return code to indicate dimension check error in subscripting a dummy array).

9. BR 14 (return to caller).

If the operator is not a multiply, the operands are the subscript and array, or the offset and symbol VMA. If the subscript/offset is a constant, it is added to the VMA of the array/symbol. If the subscript/offset is not a constant, LOADOP is called to generate the code to load the subscript/offset. GETBASE is called to assign a base register for the array/symbol VMA. The subscript/offset register which will occupy the X2 field of the instruction generated to locate or load the data, is stored in the array/symbol operand.

CZANH -- COMCON - COMBINE CONSTANTS

This routine combines two constants (nondimension) and forms a single entry in the Polish string. (See Chart AU.)

Entry

CZANH1
Via standard type-I linkage.

Routines Called

DIAGNO (CZANW1)
Issues diagnostics.

SIR Macro
Specify Interrupt Routine.

DIR Macro
Delete Interrupt Routine.

INTINQ Macro
Interrupt Inquiry.

Exit

This routine returns control to the calling program.

Operation

If both operands in the Polish string are constants, they are combined during code generation time into a single entry. If necessary, both constants are truncated or padded. Character constants are truncated or blank filled on the right. Hexadecimal constants are truncated or zero filled on the left. A low order word of zeroes is supplied for single precision constants. The low order word of double precision constants is truncated. Integer constants are always four bytes in length.

The operation to be performed is identified in the operator entry by a four-bit code corresponding to the rightmost hexadecimal digit in the instruction code.

Code	Operation
E	Logical Add
A	Add
C	Multiply
D	Divide
4	And
6	Or
9	Arithmetic relational
5	Logical relational

Except for a logical relational operator between two nonloadable operands, both operands are loaded into registers, and the RR form of the operand is executed.

The relational operations, the mask to be used in the conditional branch has been set in the operator entry. The result operand will be formed in the Polish string in place of the X operand entry. The data length of the result is set to four or eight bytes. The result will be stored in compliment form if the unary indicator is set in the operator entry.

CZANI -- OPGEN - OPERATOR CODE GENERATOR

This routine generates the code to perform all arithmetic, logical, or relational combinations between two variables, or one variable and a constant. (See Chart AV.)

Entry

CZANI1
Via type-I linkage.

Routines Called

LOADOP (CZANT1)
Generates code to load an operand.

GETREG (CZAOD1)
Assigns a general or floating point register.

GETBASE (CZANV1)
Converts a VMA virtual storage address to a base-displacement address.

Exit

This routine returns control to the calling program.

Operation

The routine is entered with a pointer to two operands and an operator. The operation to be performed is identified in the operator entry by a four-bit code corresponding to the rightmost hexadecimal digit in the appropriate instruction code.

Code	Operation
E	Logical Add
A	Add
C	Multiply
D	Divide
4	And
6	Or
9	Arithmetic relational
5	Logical relational

The leftmost hexadecimal digit is selected for the instruction code, based on the type, length, and location of the second operand.

Type	Length	Location	Hex Digit
INTEGER	4	GENERAL REGISTER	1
REAL	4	F.P. REGISTER	3
REAL	8	F.P. REGISTER	2
INTEGER	2	STORAGE	4
INTEGER	4	STORAGE	5
REAL	4	STORAGE	7
REAL	8	STORAGE	6

For relational operations, the mask to be used in the conditional branch has been set in the operator entry. The operators and masks are:

Mask	Definition	Operator
1100	Greater than	>
0100	Greater than or equal	>=
1010	Less than	<
0010	Less than or equal	<=
0110	Equal	=
1000	Not equal	¬=
0010	Not greater than	¬>
0100	Not less than	¬<

The resultant operand will be formed in the Polish string in place of the X operand entry. The data length of the result is set to four or eight

bytes. The "in register" flag is set. All registers used in the generated code will be released, except for the final result register. The generated code pointer is updated to reflect the code added to the generated code page.

Addition: The first operand is selected. If either operand is in a register, no code is generated. If Y only is in a register, the operands are exchanged. If neither operand is in a register, LOADOP is called to load the operand with the shorter data length.

If the arithmetic unary indicator of the operands do not agree, the hexadecimal digit representing the op code is changed to subtraction. The arithmetic unary indicator of the result is set if the X operand and the operator arithmetic unary indicators do not agree.

The Y operand is examined and, based on its type, length, and location, the appropriate add instruction is generated.

Multiplication: The first operand is selected. If either operand is in a register, no code is generated. If Y only is in a register, the operands are exchanged. If neither operand is in a register, LOADOP is called to load the shorter operand.

The arithmetic unary indicator of the result is set to the Exclusive OR of the arithmetic unary indicators of the two operands and the operator.

The Y operand is examined and, based on its type, length, and location, the appropriate multiply instruction is generated.

Division: If the dividend (operand X) is not in a register, LOADOP is called to generate code to load it. If the data type is integer, code is generated to load the data into the even register of the pair and to propagate the sign by shifting the operand into the odd register.

The arithmetic unary indicator of the result is set to the exclusive OR of the arithmetic unary indicators of the two operands and the operator.

The Y operand is examined and, based on its type, length, and location, the appropriate divide instruction is generated. Integer divisors with lengths of one or two bytes will be loaded via LOADOP, and a divide register instruction will be generated.

AND/OR: The first operand is selected. If either operand is in a register, no code is generated. If only Y is in a register, the operands are exchanged. If neither is in a register, LOADOP is called to generate code to load the shorter operand.

If the logical unary indicator of the operand selected is set, code is generated to invert the operand. This is accomplished by generating a load address and an RR exclusive OR, which, when executed, will invert the low-order bit of the operand.

If the logical unary indicator of the second operand is set, code is generated to invert the operand, as described above.

Based on the type, length, and location of the Y operand, an RR or RX instruction is generated to perform the AND or OR operation.

The logical unary indicator of the result is set to the Exclusive OR of the logical unary indicator of the X operand and the operator.

Arithmetic Relational: If the first operand is not in a register, a call is made to LOADOP to generate the code to load it.

If the arithmetic unary indicator of the second operand is set, the arithmetic unary indicator of the first operand is inverted, and the alternate mask for the comparison is selected. This is done to eliminate the command to load the second operand, if the operand must be complemented before performing the comparison. For example, A GT -B becomes -A LT B. The alternate masks are:

Mask	Comparison
1010	Greater than >
0010	Greater than or equal to >=
1100	Less than <
0100	Less than or equal to <=
0110	Equal to =
1000	Not equal to ¬=

If the arithmetic unary indicator of the first operand is set, an LCR instruction is generated.

GETREG is called to assign a general register which will be used for the result. Code is generated (an SR instruction) to load a logical false indicator. Logical false is zero; logical true is nonzero.

The Y operand is examined and, based on its type, length, and location, the appropriate compare instruction is generated.

The BC instruction is generated. If the logical unary indicators of the first operand and the operator do not agree, the mask is inverted before being inserted in the M1 field of the instruction. A displacement is computed to branch around the next instruction.

An LA R1,1(0,0) is generated to load a logical true indication in the result register.

Logical Relationals: If the "not loadable in general register" flag is off in the Polish string header entry, logical relationals will be processed in the same manner as arithmetic relationals. The compare instruction generated will be a logical compare, using the general registers. A logical relational between two operands which are not loadable will result in a storage-to-storage comparison of the two operands. Code is generated to load a logical false indicator into the result register. GETBASE is called for each operand to assign a base register and displacement. The CLC instruction is generated, followed by a BC instruction. If the logical unary indicators of the first operand and the operator do not agree, the mask is inverted before being inserted into the M1 field of the BC instruction. A displacement is computed to branch around the next instruction. An LA R1,1(0,0) is generated to load a logical true indicator into the result register.

CZANT -- LOADOP - LOAD OPERAND

This routine generates the code to load an operand into a general or floating point register. (See Chart AW.)

Entry

CZANT1
Via a standard type-I linkage.

Routines Called

GETBASE (CZANV1)
Converts a VMA to base-displacement address.

GETREG (CZAOD1)
Assigns a general or floating point register.

Exit

This routine returns control to the calling program.

Operation

A call is made to GETBASE to provide a base register for the data. GETREG is called to provide a pair of data registers for the operand. The load instruction is generated based on the type and length of the operand.

Type	Length	Instructions Generated
Integer	1	SR & IC
Integer	2	LH
Integer	4	L
Real	4	load short *
Real	8	load long

* If the Polish string header entry indicates that the result is to be double precision, the load short instruction will be preceded by an SR to clear the register.

The R1 field is set to the register supplied from GETREG. The B2 and D2 fields are set to GETBASE output parameters.

CZANV -- GETBASE - BASE REGISTER ASSIGNMENT

This routine assigns a base register for referencing an operand and generates code to load the base register, if necessary. (See Chart AX.)

Entry

CZANV1
Via a standard type-I linkage.

Routines Called

GETPAGE (CZANZ1)
Obtains a page for generated code.

GETREG (CZAOD1)
Assigns a general register.

Exit

This routine returns control to the calling program.

Operation

Three groups of base registers are used to reference operands in generated code. The first group (registers 10, 11, and 12) references variables; the second (register 13) references a subexpression that was stored in a PSECT by generated code; the third (register 15) references constants that are stored in the same page as the generated code.

Variables - The first time a variable is referenced, an LM instruction is generated. The R1 and R3 fields are set to 10 and 12, respectively. The location of the LM is saved. The pointer to the next available storage area for generated code is updated. The count of bases assigned is set to one. The VMA of the variable is rounded to the lower page boundary and stored as the first base assigned. It will be loaded into register 10 which will be the base register during the execution of the generated code.

For subsequent variable references, a check is made to see if the variable can be referenced by using a previously assigned base. If a previous base can be used, it becomes the base register, and the displacement is set to the difference between the variable's VMA and the base. This will minimize the number of bases and LOAD multiple commands. If a previous base cannot be used, the count of bases assigned is incremented and the variable's VMA is assigned a base as described above.

The register that will contain the base is returned as the base register with the appropriate displacement. When three base registers have been assigned, and they cannot be used in referencing the next variable, the registers are stored in the current page. The pointer to the next available storage areas for data is updated. The B2 field of the LM instruction is set to 15, and the D2 field is set to the difference between the assigned base storage and the starting address of the generated code. The base for the next variable is then assigned as if it were the first base.

Dimension factors are treated as special variables; they are contained in the array ISD symbol entry. The base register assigned for referencing these variables contains the VMA of the array ISD entry. This VMA is returned to the generated code caller if a dimension error occurs in the dynamic evaluation of a subscript. A check is made to see if a base register has been assigned for referencing the ISD entry. If one has been assigned, it is used; otherwise, the next available base register is assigned in the manner described previously. The assigned base register number is saved for SUBGEN usage. This number is returned to the GETBASE caller and the displacement is set to the difference between the dimension factor's VMA and the ISD entry's VMA.

If the operand is a FORTRAN dummy variable, the VMA just processed is the address of the adcon for the argument. GETREG is called to assign a general register for the adcon. The following command is generated: L R1, D2(0,B2) The R1 field is the register supplied by GETREG.

The following instructions are then generated:

(1)	LTR	R1,R1	(test adcon register)
(2)	BC	7,D2(B2)	(branch past the next two instructions)
(3)	LA	15,12(0,0)	(load return code to indicate a reference to a dummy variable which has not been established)
(4)	BR	14	(return to caller)

1. FORTRAN dummy variables as well as operands specified as symbols with offset are automatically aligned. For all other operands, the displacement of the operand is checked against the data length to ascertain alignment requirements. ALR R1,R2 (add base register to offset register instruction is generated if alignment is required

and double indexing is specified. Thus, the offset register becomes the base register. Also, if operand alignment is requested by the caller, an MVC D1(L,13),D2(B2) instruction is generated to move the data to aligned storage in the PCS PSECT. Register 13 is the base register for the aligned storage and the D1 field of the move instruction is set to the next aligned storage location.

2. Stored subexpressions - If the item is not identified as a constant, and the stored flag is set, the item is a stored subexpression. The stored flag is cleared. Register 13 becomes the base register while the displacement is picked up up from the item. This displacement was stored in the item when the code to store the subexpression was generated. The pointer to the next available PSECT storage location is updated, to release the subexpression storage.

3. Constants - If the item is identified as a constant, the constant is stored as data in the current page. Prior to being stored, the constant is padded or truncated as necessary. Character and real constants are truncated on the right; others are truncated on the left. Character constants are blank padded on the right. Real constants are zero filled on the right. Others are zero filled on the left. The pointer to the next available data storage location is updated. Register 15 becomes the base register. The displacement is set to the difference between the data storage and the starting address of generated code.

CZANW -- DIAGNO - ISSUE DIAGNOSTICS

This routine forms and issues diagnostics for PCS. (See Chart AY.)

Entry

CZANW1
Via standard type-I linkage.

Routines Called

PRMPT
Writes a line to a terminal.

GETCHAR (CZAMQ2)
Gets the next character.

Exit

This routine returns control to the caller.

Operation

A diagnostic message consists of two sets of characters. The first set consists of that portion of the search list string which caused the diagnostic. This set is contained in the PCS diagnostic buffer. The second set is the text of the diagnostic message and is contained in SYSMLF. Each diagnostic is assigned a unique code number, which is supplied in the diagnostic code storage area and is used to locate the diagnostic test and level.

The four diagnostic levels are:

1. Null: These diagnostics are informational only. They do not result in prompting or rejection.

2. Warning: These diagnostics are informational. They cause the user to be prompted for acceptance.

3. Operand Fatal: These diagnostics indicate that an operand is being ignored. They cause prompting or rejection.

4. Statement Fatal: These diagnostics cause the entire statement to be rejected.

The diagnostic code storage area is checked, and if it is clear, processing continues with the diagnostic scan check. If the storage area is not zero, it is assumed that it contains one or more diagnostic codes, and a diagnostic will be issued for each. If more than one diagnostic is issued, the offending portion of the source list string is contained in the first diagnostic. Subsequent diagnostics will contain the text alone.

When all diagnostics have been issued, the diagnostic scan indicator is checked. If the indicator is clear, the routine exits. If the indicator is set, the source list input has violated PCS syntax rules and the character string in the PCS diagnostic buffer is scanned. Each character is inspected. An end-of-block indicator causes an automatic termination of the diagnostic scan, regardless of syntax checks. If

a quote is encountered, the subsequent characters are checked for the terminal quote. If a left parenthesis is encountered, a parenthesis count is incremented. When a right parenthesis is encountered, the parenthesis count, if it is greater than zero, is decremented. The diagnostic scan is terminated when the parenthesis count is equal to zero and a comma or semicolon is encountered outside of the quoted string.

The diagnostic scan may result in the extraction of characters from the source list. These characters are obtained by means of the GETCHAR routine (see SCANFLD). Any character strings obtained by the diagnostic scan are not candidates for synonym substitution. When the diagnostic scan has been completed, a diagnostic for the resultant character string is issued.

CZANX -- PROMPT - USER PROMPTING

This routine issues a prompting message and solicits a user's response. (See Chart AZ.)

Entry

CZANX1
Via a standard type-I linkage.

Routines Called

PRMPT
Writes a line to the terminal and reads a response.

DIAGNO (CZANW1)
Issued diagnostics.

Exit

This routine returns to the calling program.

Operation

A prompt message consists of a message number and two character strings. The message number and the location and length of the first character string are supplied by the caller. The second character string is contained in SYSMLF and is referenced by message number. The message is written via a call to the PRMPT macro, which returns the reply. If the response is valid, it is returned to the caller. If the response is invalid, the default response is returned to the caller.

If the task is non-conversational, a diagnostic is issued in place of the PROMPT message. The diagnostic level is set to cause statement rejection.

CZANZ -- GETPAGE - ALLOCATE VIRTUAL STORAGE

This routine allocates virtual storage for PCS.

Entry

CZANZ1
Via a BASR.

Routines Called

GETMAIN
Allocates one page of virtual storage.

DIAGNO (CZANW1)
Issues a diagnostic.

Exit

This routine returns to the caller if storage is available; if not, control is passed to the caller of PCS.

Operation

GETMAIN is called to allocate one page of working storage. If the return code indicates that storage is available, return is to the caller, with the storage location in parameter register 1. If the return code indicates that storage is not available, a diagnostic is issued and GETPAGE returns control to the caller of PCS directly.

CZAOA -- VALMOD - EVALUATE MODULE NAME

This routine locates and loads the internal symbol dictionary (ISD) for the module. (See Chart BA.)

Entry

CZAOA1
Via standard type-I linkage.

Routines Called

GETMAIN Macro
Allocates storage for an ISD.

MOVEPAGE
Reads in an ISD.

FIND Macro
Locates the ISD.

HASHSEARCH (CZCDL2)
Locates the PMD for the module name with the calling sequence: CALL CZCDL2, (LIST)

where LIST consists of five parameters:

1 - Pointer to the user hash table.
2 - Zero
3 - Pointer to the symbol name.
4 - The module sequence number.
5 - The VMA of the symbol definition (on return).

Exit

This routine returns to the calling program.

Operation

The ISDMAP is first searched to determine if the ISD for the module has been loaded. If it has, the entry number in the ISDMAP is returned to the caller. If the ISD has not been loaded, the PMD for the module is located and the "ISD available" flag is checked. (See CZAMO for PMD loading procedure.) If the flag is off, an error diagnostic is formed. If it is on, the ISD is loaded and the ISDMAP is updated.

This routine may be called with one or two qualifying names. If two names are given, the search procedure described above is followed using the linkage editor qualifying name. When the ISD is loaded, a search is made through it for level 1 ISDs. When one is found, all input module names (compiler or assembler produced ISDs) are put into the ISDMAP with a pointer to the entry for the linkage editor ISD. If the linkage editor ISD has no input module with the same name as the secondary qualifying name, an error indicator is set.

If the ISDMAP already contains an entry for the names given, the routine returns with pointers to the entry or entries and performs none of the load functions described. The following error checks are made:

- No module loaded for the given name.
- Module has no ISD.
- For linkage editor modules.
  a. Two qualifying names required.
  b. Second qualifying name not found in linkage editor ISD.

CZAOB -- VALSYM - EVALUATE SYMBOL

This routine locates a symbol's entry in the internal symbol dictionary (ISD) and computes its virtual storage address. (See Chart BB.)

Entry

CZAOB1
Via standard type-I linkage.

Routines Called

HASHSEARCH (CZCDL2)
Locates the CSD for the control section containing the symbol with the calling sequence: CALL CZCDL2,(LIST)

where LIST consists of five parameters:

1 - Pointer to the user hash table.
2 - Zero
3 - Pointer to the symbol name.
4 - The module sequence number.
5 - The VMA of the symbol definition (on return)

Exit

This routine returns to the calling program.

Operation

The ISD for the primary qualifying name is located. If the ISD is for an assembled or compiled module, it is searched for the symbol entry. If, however, it is for a link edited module, the ISD for the secondary qualifying name is searched.

If a FORTRAN statement number is being processed, the statement number table of the FORTRAN ISD is searched. If a subscript value is specified, the entry must be found in the statement number table, or the data location is identified as an error. If the entry is found, the subscript value (if any) is applied to the entry in the statement number table to locate the appropriate entry. The symbol displacement in the control section is obtained from the statement number table entry. The control section name is obtained from

the next to last entry in the section name table. If the entry is not found, the data location is re-identified as an internal symbol.

If an internal symbol is being processed, the symbol table of the FORTRAN or assembler ISD is searched. If the symbol entry is not found, the data location is redefined either as an error, if an implicit or explicit internal symbol is being processed, or as null, if the symbol is not implicitly or explicitly qualified.

If the symbol table entry is found, the entry type is inspected. If the entry type is immediate data, the data location item is completed and the routine exits. If the entry type indicates that the symbol is a DSECT, the data location identification is checked to ensure that an internal symbol with offset is being processed. The data location item is then completed and the routine exits.

In all other cases, the type and length attributes are stored in the LOCITEM. The symbol displacement in the control section is obtained from the symbol entry. The control section number in the symbol entry is used to index the section name table to obtain the control section name.

If a linkage editor ISD was specified as the primary qualifier, the ISD is searched down to locate the lowest level in the defining ISD, then up to compute a displacement for the control section. To facilitate this search, a temporary two-word search list is formed pointing to the preceding ISD (word 1) and the input module heading (word 2).

A two word entry is made for each ISD level. Generation of the list continues until a level 1 ISD is obtained. The level 1 ISD is then searched to see if any input module name equals the secondary qualifying name. If no match is found, the entry for the level 1 ISD is deleted from the search list, the entry for the next higher level (i.e., the previous entry in the search list) is obtained, and the search for a level 1 ISD is continued.

When a level 1 ISD containing the secondary qualifying input module name is found, a check is made to determine if the control section name for the symbol is one of the input control sections of the input module. If the control section name is not found, the level 1 ISD is deleted from the search list and the search continues.

If the control section was an input control section in the formation of the level 1 link edited program, a search up the ISD levels (using the entries in the search list) is initiated, in order to compute the displacement in the final control section (i.e., the control section in the loaded module). If, at any point, during the upward search, a control section is not found, the downward search is resumed.

When the final control section name has been determined, the HASHSEARCH routine is called to locate the control section definition. The symbol displacement in the control section is then added to the control section VMA and stored in the LOCITEM.

CZAOD -- GETREG - REGISTER ASSIGNMENT

This routine assigns registers required for generated code. (See Chart BC.)

Entry

CZAOD1
Via a standard type-I linkage.

Routines Called

None

Exit

This routine returns control to the calling subroutine.

Operation

The routine is entered with a pointer to an operand entry in the Polish string. The register type requested is determined.

General Purpose: Five pairs of registers, in order from 0-1 through 8-9, are used for loading operands. The register assigned to the operand is stored in the operand entry and the "in register" flag is set. If a request is made for a register, and all five have been assigned, the first used registers are allocated PSECT storage, and a register pair is freed for the current call. The operand entry in the Polish string is updated to indicate that the

operand has been stored, and where it is stored. An ST instruction is added to the generated code to store the register.

Floating Point: Registers are taken in sequence. The register assigned to the operand is stored in the operand entry, and the "in register" flag is set. If a request is made, and all four registers have been assigned, the first used register is allocated PSECT storage, and the register is used for the current call. The operand entry in the Polish string is updated to indicate that the operand has been stored, and where it is stored. An STD instruction is added to the generated code to store the floating point register.

CZAPB -- PCSPUT - PCS OUTPUT CONTROL

This routine exercises overall control in Phase III for processing immediate and dynamic statements. (See Chart BD.)

Entry

CZAPB1
Is used by the task monitor to call PCS to process a PCSVC interrupt. A standard type-I linkage is used.

CZAPB2
Is used by PCS Input Phase II to process an immediate statement. A standard type-I linkage is used.

Routines Called

FINDLOC (CZAPC1)
After an SVC interrupt, locates the matching entry or finds the next available entry in the location table (LOCTAB).

LINE (CZAPH1)
Writes a line on the terminal device.

FORMDIAG (CZAPI1)
Calls PRMPT to write a diagnostic message on the terminal device.

INTERVENE (CZAMZ3)
Interrupts processing of the current user program and provides the linkage to CA&E to create a new sublist.

DISPDUMP (CZAQA1)
Processes phrase lists.

VISAM PUT
Writes a line on the PCSOUT data set.

GENCALL (CZAPN1)
Executes the generated code.

SYMGEN (CZAPG1)
Generates a symbol for a VMA.

FINDREAL (CZAPL1)
Locates the VMA of a recomposed instruction.

SAVIX (CZAPK1)
Recomposes the user's instruction which was replaced by an SVC.

USER CONTROL ROUTINE (CZAMZ1)
Initiates execution of a user's program.

EXPAND SOURCE LIST (CZASC2)
Expands the source list by one page.

Exit

PCS Output Control returns to the calling program.

Operation

This routine has two entry points. It is entered from PCS Input Phase II to process an immediate statement, and it is entered from the task monitor to process a dynamic statement.

Immediate Statement Entry: The location of a statement table entry (STATAB) is supplied as a parameter. The immediate statement switch is turned on, and control is given to the common processing routine.

Dynamic Statement Entry: Control is received from the task monitor via a standard type-I linkage. However, no parameters are supplied. The interrupt address is picked up from the VPSW in the interrupt storage area (ISA1). FINDLOC is entered with the VMA of the interrupt. If the LOCTAB entry was found, a check of the entry type is made.

If the type is a RETURN, the interrupt was caused by the ENTER PCSVC that was placed following the recomposed user's instruction. If the recomposed instruction is a NOPR instruction, the overlaid instruction was a branch type instruction and must be interpreted. FINDLOC is called to locate the LOCTAB entry for the original instruction.

The branch instruction is then interpreted to determine if the branch is successful. If it is, the user's VPSW is modified to transfer control to the branch address. If it is unsuccessful, the user's VPSW is modified to point to the next sequential instruction in his program. The entry is then cleared from the location table.

If the LOCTAB entry was not a RETURN, the pointer to the STATAB entry is picked up, and control is given to the common processing routine.

If a LOCTAB entry for the interrupt address was not found, it is classified as an illegal entry; this is probably caused by an erroneous SVC instruction or by the user who executed an SVC via an EX instruction. A diagnostic is issued, indicating the contents of the VPSW and of the erroneous instruction. Control is then passed to INTERVENE to halt execution of the user's program, create a new sublist, and obtain the next line of input.

Common Processing for Immediate and Dynamic Statements

The STATAB entry is located and the dynamic count is incremented. The phrase list is then located and a branch to the appropriate routine is made based on the phrase list identification.

AT: Each entry in the AT list is checked to see if the entry is for the current interrupt address. If an entry is found for the current interrupt address, the address of the next STATAB entry is set from the AT list entry. Processing continues with the identification of the next phrase list when an entry is found or when all entries have been inspected.

IF: GENCALL is called to execute generated code. If the result is true, processing continues with the identification of the next phrase list. If the result is false, and a dynamic statement is in process, processing continues with the next STATAB entry. CA&E is notified of a false immediate IF phrase.

DUMP: The standard header is written if necessary, to the PCSOUT data set for dynamic DUMP phrases. Processing of the DUMP phrase list is identical with the processing of a DISPLAY or SET phrase list.

DISPLAY and SET: DISPLAY/DUMP is called to process the phrase list. Processing continues with identification of the next phrase list.

STOP: If LIMEN is defaulted to "I", a STOP followed by the standard header is written out. If LIMEN is not defaulted to "I", only the statement number is written out. If a statement number has already been written, the STOP indicator is set and processing continues with the next STATAB entry.

GO: If LIMEN is defaulted to "I", the user is notified as to the symbolic instruction where program execution is resumed. The BRANCH/GO indicator is set. Processing continues with the next STATAB entry.

BRANCH: If necessary, the standard header is written on the SYSOUT device for dynamic statements. If the branch address is offset, GENCALL is called to execute the generated code and the result is added to the branch address. The user is notified as to the symbolic instruction where program execution is resumed. The BRANCH/GO indicator is set and processing continues with the next STATAB entry.

CALL: If a parameter list is to be constructed, GENCALL is called to execute generated code to evaluate each parameter. If the parameter is returned in a register, the parameter is inserted in the source list. The address of each parameter is stored in the parameter list.

The V-con, R-con, and the parameter list (which may be null) are inserted in the source list. If the statement is immediate, processing continues with the next STATAB entry. If the statement is dynamic, the user control routine is called to initiate program execution. When control returns, processing continues with the next phrase list.

CONTINUE: Processing continues with the next phrase list.

TERMINATE: Processing continues with the next STATAB entry.

Processing of Next STATAB Entry: If the statement is immediate, control returns to the caller. CA&E is notified of an end-of-level if a BRANCH or a GO phrase was processed.

For dynamic statements, the standard header is written, if necessary. If there is a next STATAB, and a BRANCH or STOP phrase was processed, a diagnostic containing the unprocessed statement number is issued and execution of the user program is halted. If there is a next STATAB entry, and a BRANCH or STOP phrase was not processed, the next STATAB entry is processed as described above.

When all STATAB entries have been processed, the user's instruction is recomposed if necessary. If the user program is to be halted, INTERVENE is called and when control returns, PCSPUT locates the save area pointed to by CZAMZ4, and issues a return. Control returns to the caller if the user's program is not halted.

CZAPC -- FINDLOC - LOCATION TABLE SCAN

This routine hashes a VMA and finds an appropriate location table (LOCTAB) entry. (See Chart BE.)

Entry

CZAPC1
Via a standard type-I linkage with a VMA stored in the PSECT.

Routines Called

GETPAGE
Allocates a page for LOCTAB.

Exit

This routine returns to the calling program with a return code:

0 - Available entry found.

4 - Matching entry found.

Operation

FINDLOC is called for two purposes: to locate an available entry in LOCTAB, or to locate a matching entry. The location table is searched by hashing the VMA until either an available (null) entry or a matching entry is found. Entries that are unlinked by REMOVE are considered available if a matching entry is not found.

FINDLOC takes the virtual memory address provided, hashes the address, and uses the hash result to search the location table. The formula for hashing is:

1. Divide VMA by constant X'EEEF'. Save remainder as hash.
2. Take nine low-order bits from hash.
3. If the result is more than 340, subtract 171.
4. Multiply by 12 to get byte address.
5. Combine byte address with LOCTAB page address to get address of LOCTAB entry.
6. To get next address, shift off low-order bit of hash and continue with next nine bits. Eight combinations are possible.

If a LOCTAB entry is found to be null, its address is returned as an available entry. If a non-null entry is found, the SVC location in the entry and the VMA that was hashed are compared. If they are equal, the LOCTAB address is returned as a matching entry. If they are not equal, the first LOCTAB entry identified as REMOVE is set as the null entry. Then the search continues.

After all entries for a page of the table have been searched, the overflow pointer is examined.

If it is nonzero, the search is continued on the new page. After all pages have been checked (i.e., the overflow pointer on the last page is zero), the null entry is checked. If a LOCTAB entry identifies as REMOVE was found, the address of the entry is returned as the available entry. If all LOCTAB entries inspected were "in use," a new page is allocated via GETPAGE. The address of the new page is stored in the overflow pointer of the last page.

If all existing pages of the location table were examined without finding an entry, the return parameter is set to zero before returning to the calling program.

CZAPG -- SYMGEN - SYMBOL GENERATOR

This routine converts a VMA of a user instruction to symbolic form for display purposes. (See Chart BF.)

Entry

CZAPG1
Via a standard type-I linkage.

Routines Called

MAPSEARCH (CZCCQ)
Locates the control section information (CSD) for the VMA with the calling sequence: CALL CZCCQ, (LIST) where LIST consists of three parameters:

1 - VMA to be resolved
2 - Function code (set to zero)
3 - Pointer to CSD (upon return)

Exit

This routine returns to the calling program.

Operation

The routine searches all available ISDs and PMDs and attempts to reconstruct the symbolic address, as it was defined in the FORTRAN or assembly program, into the form:

FORTRAN - QNAME.99(ddd)

Assembly - QNAME.LNAME.(offset)

where QNAME is a qualifying program module name

99 is the nearest preceding statement number that can be followed by a subscript

(ddd) is the count of executable FORTRAN statements

LNAME is the nearest preceding internal symbol which may be followed by (offset)

(offset) is the offset interval in bytes

If a link-edited module is being processed, two qualifying names will be returned. The first is the name of the load module; the second is the assembler or compiler module name that contains the internal symbol.

If the location occurs in the control section, before any internal symbols or statement numbers, or if the control section contains none, the symbol has the same form, except that LNAME is the control section name and the statement number designation (99) is zero or omitted.

The MAPSEARCH routine is called to locate the entry that brackets the user's VMA. The entry points to a PMD from which the module and control sections names are obtained. If the PMD indicates the module has an ISD, the ISDMAP is searched. If a matching entry is found, the ISD type is checked. The type may be:

- Assembler ISD - The section name entries are scanned, to find the correct section. All symbol entries for the control section are examined, to locate the closest preceding internal symbol.
- FORTRAN ISD - The statement number entries are examined, to locate the closest preceding statement number.
- Linkage editor ISD - Each link-edit operation generates a new ISD, and links together all previous, associated ISDs.

The qualifying name (QNAME) is saved as the first of two qualifying names to be displayed. The control section name pointed to by the memory map is used as an argument, to search the link-edit produced output control section.

When a match is found, the input module name containing the control section is saved as the current QNAME. A new value of LNAME is determined, depending on whether input control sections processed by the linkage editor are unchanged, renames, or combined. The ISD for the input module is obtained, and the type checked. If it is a link-edit ISD, the procedure described in this paragraph is repeated, using the new values of QNAME and LNAME.

If the ISD is an assembler or FORTRAN ISD, processing is as described above. When the link-edit ISD search is completed, QNAME contains the module name while LNAME contains the control section name.

If a list is exhausted without a match, processing is discontinued, and the symbol is resolved as an external symbol. If an external symbol is to be generated, the external symbol used is the control section name containing the

VMA. The offset from the control section name is expressed as a hexadecimal value.

CZAPH -- LINE

This routine sends a message line to the user.

Entry

CZAPH1
Via a standard type-I linkage.

Routines Called

GATWR
Sends a line to the user.

Exit

This routine returns to the calling program.

Operation

A character string has been formatted in the line area. The ending pointer will determine the number of characters in the message. A call is made to GATWR to write the message and the message area is then cleared.

CZAPI -- FORMDIAG - FORMAT DIAGNOSTIC

This routine sends a diagnostic message to the user.

Entry

CZAPI1
Via a standard type-I linkage.

Routines Called

None

Exit

This routine returns to the calling module.

Operation

A diagnostic code is returned and used to search a table for the proper message identifier. This identifier is used to select a message from the System Message file. If the message requires an insert, a pointer to the insert is placed in a register. The PRMPT macro is issued to write the message to the user's terminal. Upon return from PRMPT this routine clears the line buffer used to hold message inserts and returns to its caller.

CZAPK -- SAVIX - SAVED INSTRUCTION EXECUTION

This routine recomposes the machine instruction in the user's program that Phase II control replaced with an SVC. (See Chart BG.)

Entry

CZAPK1
Via a direct branch.

Routines Called

FINDLOC (CZAPC1)
Finds an available entry in LOCTAB.

LINE (CZAPH1)
Issues a diagnostic line.

Exit

This routine returns to PCSPUT via a direct branch to PB420.

Operation

The VMA immediately following the SVC in the user's program and the first two bytes of the original instruction are found in LOCTAB. The instruction's operation code is checked to see if the instruction could result in a branch (is a branch type instruction). If it could, a NOPR instruction is substituted; if not, the instruction is recomposed in working storage and followed by an ENTER PCSVC.

When the subject instruction of an EXECUTE is of the branch type, a NOPR replaces the EXECUTE, so that the branch can be interpreted after the RETURN.

A RETURN entry is placed in LOCTAB for the ENTER PCSVC. The address of the next sequential instruction in the user's program is stored in the entry along with the length of the recomposed instruction.

CZAPL -- FINDREAL - FIND REAL ADDRESS

The purpose of this routine is to determine if a location contains an instruction recomposed by PCS.

Entry

CZAPL1
Via a standard type-I linkage.

Input Parameters: Register 1 contains the virtual storage address.

Output Parameters: Register 15 contains either the address of the original instruction or zero for a non-recomposed instruction.

Routines Called

FINDLOC (CZAPC1)
Finds location table entry.

Exit

The routine returns to the calling program.

Operation

The VMA is compared to the address of the page containing the recomposed instructions. If the VMA is in the recomposed instruction page, FINDLOC is called to search the location table for the matching entry. The address of the instruction that was overlayed by the PCSVC is computed from the return address and delta in the RETURN LOCTAB entry. The caller is then returned to the VMA of the original instruction that was overlayed by the PCSVC.

CZAPN -- GENCALL - CALL GENERATED CODE

This routine is entered with a pointer to generated code. It executes the code and stores the results in CZAMA9. (See Chart BH.)

Entry

CZAPN1
Via a standard type-I linkage.

Routines Called

SIR Macro
Specify Interrupt Routine

DIR Macro
Delete Interrupt Routine

INTINQ Macro
Interrupt Inquiry

LINE
Issues diagnostic line

Exit

This routine returns to the calling program. The CZAMA9 area contains three parameters:

1 - The return code from generated code. If an error occurred during the execution of the generated code, the return code is set to 12.
2 - Not used.
3 - General register 1, as a result of executing the generated code.

Operation

The SIR Macro is executed. Control is passed to the generated code. The results of the generated code are stored in CZAMA9. The DIR Macro is executed. The INTINQ Macro is executed and if it detects an interrupt, an error return code is created.

If the generated code execution did not cause an interrupt, the return code is checked.

If the return code indicates an error, the standard header is written if necessary. Diagnostics identifying the error and the phrase being processed are issued and an error return code of 12 results.

CZAQA -- DISPDUMP - DISPLAY/DUMP CONTROL

This routine controls the processing of a DISPLAY, DUMP, or SET phrase list. (See Chart BJ.)

Entry

CZAQA1
Is entered from PCSPUT via a standard type-I linkage, with the following two-word parameter list:

Word 1 - Location of the first phrase list

Word 2 - Base location of the ISDMAP

Routines Called

DISOUT (CZAQU1)
Writes a qualification line.

NEXTLIST (CZAQB1)
Processes a phrase list.

GDV Macro
Gets the default value for LINEMENT.

Exit

This routine returns to the caller.

Operation

DISPDUMP processes the phrase list header. If the header indicates the presence of a qualification name, and if this name is not the same as the last name, a line of the form: QUALIFICATION IS BY...is issued to the terminal.

The default value for LINEMENT is obtained if a SET phrase list is being processed.

NEXTLIST is called to process the entries in the phrase list. When control returns, if an error or an attention interrupt has occurred, the return code is set to 4 (error/stop). Otherwise, the return code is set to 0 (continue).

CZAQB -- NEXTLIST - PROCESS PARAMETER LIST

This routine processes each item in a SET, DUMP, and DISPLAY phrase list, and initializes the display list (DISPLIST) for each entry. (See Chart BK.)

Entry

CZAQB1
Via a standard type-I linkage.

Routines Called

GENCALL (CZAPN1)
Executes generated code.

DISREG (CZAQF1)
Displays registers.

NEXTISD (CZAQD1)
Sets up display list item.

NEXTITEM (CZAQC1)
Processes DISPLIST.

CKCLS Macro
Checks storage protection class.

DISRHEAD (CZAQQ1)
Formats and writes range headers and issues diagnostics.

MAPSEARCH
Locates the CSD for a VMA with the calling sequence: CALL CZCCQ, (LIST) where LIST consists of three parameters:

1 - The VMA to be resolved.
2 - A function code of zero.
3 - Contains the CSD address (on return).

Exit

This routine returns to the calling program.

Operation

The phrase list entry header is processed and the display list (DISPLIST) items are initialized and a branch based on the entry identification is executed.

Register: DISREG is called to perform all necessary processing.

Dynamic Count: The DISPLIST item for the dynamic count is formed.

External Symbol: The DISPLIST item for an external symbol is formed. If an offset is specified, GENCALL is called to evaluate the offset. If a range is in process, a second display list item is formed for the range upper limit. If a range is not in process, and an offset is not specified, the DISPLIST item is adjusted if the external symbol is a module name or a control section name.

Internal Symbol, Statement Number, Subscripted Array: The DISPLIST item for an internal symbol is formed. NEXTISD is called to aid in the formation of the item. GENCALL is called to evaluate the subscript/offset. If a range is in process, a second DISPLIST item is formed for the upper limit of the range.

Hexadecimal Address: The DISPLIST item is formed for the hexadecimal address. A second DISPLIST item is formed for a range.

Expression: The display list item for an expression is formed.

Command Variable: The display list item for a command variable is formed.

When the DISPLIST items have been formed, DISRHEAD is called to form a range header if necessary. If an offset was processed in creating a DISPLIST item, a storage class check is made to insure all storage is assigned and to

test for read only, or no read/no write storage. If a SET phrase is being processed, GENCALL is called and the result is stored in the specified area. If a positive response is requested, the result will be displayed. If necessary, the range header is written. NEXTITEM is called to perform the formatting and output functions. The next entry in the phrase is then processed.

CZAQC -- NEXTITEM - PROCESS DISPLAY LIST

This routine processes the display list (DISPLIST) generated by NEXTLIST. It determines which routine to call to convert the data item, and the order of call when a range is specified. (See Chart BL.)

Entry

CZAQC1
Via a standard type-I linkage.

Routines Called

SIMVAR (CZAQG1)
Displays a simple variable.

DISARAY (CZAQJ1)
Displays an array.

DISHEX (CZAQM1)
Displays a hexadecimal range.

NEXTISD (CZAQD1)
Resets display list items.

DBIN (CZAQT1)
Display a variable or range in binary

DISOUT (CZAQU1)
Writes a line.

EBCDTIME Macro
Converts Version ID.

DISYM (CZAQR1)
Formats a symbol name.

Exit

This routine returns to the calling program.

Operation

The IDENT of each entry is examined, and a call to the appropriate routine is made to display either the full range or a portion of the range. If the full range was not displayed, NEXTISD is called to reset the first DISPLIST item.

CZAQD -- NEXTISD - PROCESS NEXT ISD ENTRY

This routine sets entries in the display list (DISPLIST) from the ISD entry, and determines a type and length for data fields between two ISD entries. (See Chart BM.)

Entry

CZAQD1
Entry is via a standard type-I linkage.

Routines Called

None

Exit

This routine returns to the calling program.

Operation

When the routine is entered for the first item in the range, the offset (displacement from the control section) is moved from the ISD to DTEMPLOC. This will be used to control the processing of items through the ISD. After the first time, the next unprocessed ISD entry is examined. The displacement of this item is compared with the updated DTEMPLOC, to determine whether there are intervening undefined areas.

If the ISD entry is the next item to be displayed, the item display list is set, based on the ISD information. Length, type, and subscript dimensions are placed in the list, and a flag is set if the item is to be displayed in the assembler DC format.

If the ISD entry is not the next item to be processed, the information for the display list is generated. A test is made on the item to determine if it is an instruction, or if it is between two instructions. Based on this test, the list will be set to display the item as:

1. An instruction, if the previous item was an instruction, and the byte at the VML contains a legal operation code.

2. A hexadecimal field displayed in the assembler DC format; e.g., X'FFFFFFFF'.

3. A character string displayed in the assembler DC format; e.g., 'ABCDE'.

4. A hexadecimal range.

When all ISD entries for a control section have been processed, the remaining area to be displayed is specified in the list to be formatted as a hexadecimal range.

CZAQF -- DISREG - DISPLAY REGISTERS

This routine displays one or more general purpose registers, or floating point single or double precision registers. (See Chart BN.)

Entry

CZAQF1
Via a standard type-I linkage.

Routines Called

GENCALL (CZAPN1)
Calls generated code.

ADDITEM (CZAQH1)
Converts and moves an item to the line.

DISOUT (CZAQU1)
Writes a line.

Exit

This routine returns to the calling program.

Operation

The display list (DISPLIST) is initialized, based on the register type involved. If a SET phrase is being processed, GENCALL is called. If an error is detected upon return, the "error stop" flag is set. No further action is performed. If an error is not indicated, the data is moved to the proper register in the interrupt storage area. If positive response is suppressed (LINEMENT $\neq$ I), a return is made. If positive response is required, processing continues.

The register numbers are placed on the line. ADDITEM is called to convert and move one register to the line. The line is output when all registers requested have been converted, or when the output line is full.

CZAQG -- SIMVAR - DISPLAY SIMPLE VARIABLE

This routine formats and outputs a simple variable. (See Chart BO.)

Entry

CZAQG1
Via a standard type-I linkage.

Routines Called

DISINST (CZAQI1)
Formats and displays an instruction.

DISYM (CZAQR1)
Formats symbol name.

ADDITEM (CZAQH1)
Converts an item and moves to line.

DISOUT (CZAQU1)
Writes a line.

Exit

This routine returns to the calling program.

Operation

If the item to be displayed is an instruction, a call is made to DISINST. If a variable is to be displayed in assembler format, the data field is converted first to hexadecimal characters and placed in the line. The item is then converted according to data type via ADDITEM, and moved to the line in place of "operands."

For a simple variable, a call is made to DISYM, which formats and places the symbol on the line. The equal sign is then moved to the line, followed by the item converted by ADDITEM. DISOUT is called to write the line on the terminal or data set.

CZAQH -- ADDITEM - CONVERT AN ITEM BY DATA TYPE

The routine converts the item according to its type and moves it to the line image. (See Chart BP.)

Entry

CZAQH1
Via a standard type-I linkage.

Routines Called

REALCON (CZAQV1)
Converts a real number.

DISOUT (CZAQU1)
Writes a line.

Exit

This routine returns to the calling program.

Operation

The proper conversion routine is selected according to data type. This may require converting an item in a work area prior to its being moved to the line. The conversion routines are:

Integer: The item is converted to decimal and unpacked into a hold area, preceded by the sign. Leading zeros are suppressed, unless the item is a member of an array.

Real: The item is converted to the standard FORTRAN format with specification of E15.8 (single precision) and D23.16 (double precision). A call is made to REALCON to make the conversion and place the result in a hold area.

Complex: Two calls are made to REALCON to convert both parts of the number and place the result in a hold area.

Logical: If a variable is not zero, the constant .TRUE. is moved to the hold area; otherwise, .FALSE. is moved. If the item is a member of an array, these will be abbreviated to .T. and .F., respectively.

Immediate: Converted as a fullword integer.

Character: The characters need no conversation, and are moved directly to the line.

Address: Converted to hexadecimal characters.

Data types other than the ones listed will be converted to hexadecimal characters and stored for output by words.

After conversion, the item is moved to the line. If all the characters cannot fit on the line, the line is filled and written, and a new line begun.

CZAQI -- DISINST - DISPLAY INSTRUCTION

The routine formats and outputs an instruction in the assembler format. (See Chart BQ.)

Entry

CZAQI1
Via a standard type-I linkage.

Routines Called

DISOUT (CZAQU1)
Writes a line.

FINDLOC (CZAPC1)
Finds the LOCTAB entry containing the original overlayed instruction.

Exit

This routine returns to the calling program.

Operation

A header line is displayed prior to the first instruction. If the location contains a PCSVC, FINDLOC is called. The original instruction then replaces the SVC for formatting. The location is converted to hexadecimal digits and placed in the line. Each halfword of the instruction is then converted to hexadecimal. A table lookup gets the mnemonic operation code. The proper format for the operands is selected from the OPC table, and the operands are added to the line in accordance with standard assembler conventions.

```
RR instructions - R1, R2
                  SPM - R1
                  SVC - I

RX instructions - R1, D2 (X2, B2)

RS instructions - R1, R3, D2 (X2,B2)
shift instructions -  R1, D2 (B2)

SI instructions - D1 (B1), I2
                  or - D1 (B1)

SS instructions - D1 (L, B1), D2
                        (B2)
                  or - D1 (L1, B1), D2
                        (L2, B2)
```

If the instruction has an ISD entry, the symbol is also placed on the line. For a FORTRAN program, this might involve counting backwards to the first nonzero statement number.

If a base register is indicated in the operand of an assembler program instruction, a search of the ISD's Using Table that covers that location is made. If the register definition is found, the register contents and the instruction's displacement are added and used to scan the ISD for a symbol entry. If a symbol entry is found, it is placed in the output line next to the operands.

CZAQJ -- DISARAY - DISPLAY ARRAY

This routine determines whether or not a line of an array is to be suppressed or displayed. (See Chart BR.)

Entry

CZAQJ1
Via a standard type-I linkage.

Routines Called

DISALINE (CZAQK1)
Formats a line of array elements.

ADDITEM (CZAQH1)
Converts and moves an item to the line.

DISOUT (CZAQU1)
Writes a line.

DISYM (CZAQR1)
Formats a symbol name.

Exit

This routine returns to the calling program.

Operation

The maximum length of an item, after conversion, is computed in order to determine the number of items that will fit on one line. For a one-dimensional array, the number on a line will depend on the length of the line. Otherwise, it will depend on the dimension of the first subscript.

Each element of the array is compared with the next, and a count is maintained of the number of equals. If all the items on two or more lines are equal, a line is displayed indicating the beginning and ending subscript values, and the value of the item that was suppressed. When a line is not suppressed, a call is made to DISALINE to format and write the line.

CZAQK -- DISALINE - DISPLAY A LINE OF AN ARRAY

This routine formats and outputs a line of an array. (See Chart BS.)

Entry

CZAQK1
Via a standard type-I linkage.

Routines Called

ADDITEM (CZAQH1)
Converts an item and moves to the line.

DISOUT (CZAQU1)
Writes a line.

Exit

This routine returns to the calling program.

Operation

The items are arranged on the line, in the order of the value of the first subscript. If the value is not a multiple of the number of items on a line, it is shifted in the line image to its proper position.

Each item in the line is converted, according to data type, by means of ADDITEM, and placed in the line. The subscript values of the first item are printed to the left of the line.

The line is written when the last item in the array has been converted, the first subscript value after updating has been reset to one, or the line is full.

A "last item" flag is set when all items in the array have been displayed.

CZAQM -- DISHEX - DISPLAY A RANGE IN HEX

This routine determines whether or not a hexadecimal line is to be suppressed or displayed. (See Chart BT.)

Entry

CZAQM1
Via a standard type-I linkage.

Routines Called

SIMVAR (CZAQG1)
Displays a hexadecimal field in the assembler format.

DISHLINE (CZAQN1)
Formats a line of hexadecimal data.

DISOUT (CZAQU1)
Writes a line.

Exit

This routine returns to the calling program.

Operation

If the flag to display the hexadecimal field in the assembler format is on, the SIMVAR routine is called. Otherwise, the number of words to be formatted on one line is determined. Each word to be displayed is compared with the next, and a count is maintained of the number of equals. If all the words on two or more lines are equal, those lines are suppressed, and a line is displayed indicating the beginning and ending location and the contents of the suppressed words.

When a line is not suppressed, a call is made to DISHLINE to format and write the line.

CZAQN -- DISHLINE - DISPLAY A HEX LINE

This routine formats and writes a line of hexadecimal characters. (See Chart BU.)

Entry

CZAQN1
Via a standard type-I linkage.

Routines Called

DISOUT (CZAQU1)
Writes a line.

Exit

This routine returns to the calling program.

Operation

The beginning VMA is converted to hexadecimal and placed in the line. Since hexadecimal characters are formatted in multiples of four words to a line, the VMA of the first byte, if not a multiple of sixteen, causes the line pointer to shift to the appropriate line position representing the VMA. If only one line is being displayed, this routine is bypassed.

Each word is converted to hexadecimal characters. In the case of the first and last positions of the range, less than a full word could be converted. For DUMP, a character representation is also formed, which will appear to the right of the hexadecimal representation.

DISOUT is called when the last byte has been converted or the line becomes full. A "last item" flag is set when all the characters in the range have been displayed.

CZAQQ -- DISRHEAD - FORMAT RANGE HEADER

This routine forms a header for a range specified in a DISPLAY, DUMP, or SET command. It also retrieves location identifiers, such as a symbol or hex address, for diagnostic messages.

Entry

CZAQQ1
Via a standard type-I linkage.

Routines Called

MAPSEARCH (CZCCQ)
Finds the CSECT containing a VM address.

DISYM (CZAQR1)
Forms a symbol corresponding to a VM address.

DISOUT (CZAQU1)
Writes a range header.

DIAG (CZAQX1)
Writes a diagnostic message.

Exit

This routine returns to the calling program.

Operation

This routine calls DISYM to form a symbol or location identifier for the beginning VM address in the DISPLIST. If a range is specified, DISYM is called a second time to form the rest of the range header.

If an error code is present in DCODE, DIAG is called to write a message to the user.

If the range is specified as an external symbol, MAPSEARCH is called to find the CSECT in which it is included.

When all information is collected, DISOUT is called to write the line on the user's SYSOUT.

CZAQR -- DISYM - FORMAT SYMBOL

This routine formats a symbol or a hexadecimal location on a line. (See Chart BV.)

Entry

CZAQR1
Entry is via a standard type-I linkage.

Routines Called

None

Exit

This routine returns to the calling program.

Operation

If the IDENT indicates an external symbol, the pointer to the PMD is used to pick up the symbol name. The offset, if any, is added to the line.

If the IDENT indicates hexadecimal, the VMA is converted to hexadecimal characters, and placed on the line.

Internal symbols are preceded by the module name, unless a qualify statement is in effect. Modules that have been link-edited will also show the current module name. The offset or subscript value is added to the line.

FORTRAN statements with a zero statement number will be displayed as a subscript to the previous nonzero statement number.

The final process scans the complete symbol, including offset and subscript, and shifts the characters to remove any blanks.

CZAQT -- DBIN - FORM A LINE IN BINARY FORMAT

This routine creates a line of binary data.

Entry

CZAQT1
Via a standard type-I linkage.

Routines Called

DISYM
Forms a symbol corresponding to a virtual memory address.

DISOUT
Writes a line.

Exit

Returns to the calling program.

Operation

A line is created containing the given number of bytes in binary format, and the proper virtual memory identifier is placed in front of it.

CZAQU -- DISOUT - OUTPUT A LINE

This routine sends a line to the user via GATE for DISPLAY and SET, or via VISAM to PCSOUT for DUMP. (See Chart BW.)

Entry

CZAQU1
Via a standard type-I linkage.

Routines Called

GATWR
Writes a line on the terminal device.

VISAM PUT
Writes a line on the PCSOUT data set.

Exit

This routine has two exit points:

- To DISPDUMP if the display has been terminated by an attention

interrupt. A direct branch is made to QA040.

- To the calling program.

Operation

Before a line is sent to the terminal, the interrupt flag is checked. If the flag is on, the user has interrupted to bypass a lengthy display, and the DISPDUMP routine returns to the control module. Since the interrupt flag cannot be on unless the allow interrupt flag is also on, at least one line of data will be transmitted.

Output to the user terminal is via the GATWR routine. Dump output is put on the PCSOUT data set by the VISAM PUT macro.

CZAQV -- REALCON - REAL NUMBER CONVERSION

This routine converts a floating point number in single or double precision to the standard FORTRAN format:

Single ±.XXXXXXXX E±XX

Double ±.XXXXXXXXXXXXXXXX D±XX

(See Chart BX.)

Entry

CZAQV1
Via a standard type-I linkage.

Routines Called

None

Exit

This routine returns to the calling program.

Operation

The number is converted to a fraction with a characteristic of zero, by multiplying or dividing the number by a power of ten. The exponent corresponding to the power of ten is saved. The resulting fraction is converted to decimal digits by successively multiplying the hexadecimal number by 10. As each decimal digit shifts into the high-order position, it is zoned and moved to the next position in the output area. Leading zeros are eliminated and the exponent is adjusted. The exponent and signs are then moved to the output area along with the E/D indicator.

CZAQX -- DIAG - DYNAMIC DIAGNOSTIC

This routine forms the diagnostic for all errors detected by the DISPDUMP routine.

Entry

CZAQX1
Via a standard type-I linkage.

Routines Called

PRMPT
Writes a line on the terminal device.

Exit

This routine returns to the calling program.

Operation

A message identifier from SYSMLF is selected, based on the diagnostic code set by the calling subroutine. The keyword which indicates the reason for the diagnostic and also the action that was not taken is used as a parameter. PRMPT is called to send the line to the terminal. The error flag is set to return an error code to the caller of the DISPDUMP routine.

This section contains the flowcharts for the various PCS routines. The charts are arranged in the same order as the routine descriptions. However, not all descriptions are illustrated by a chart.

Program Logic Manual

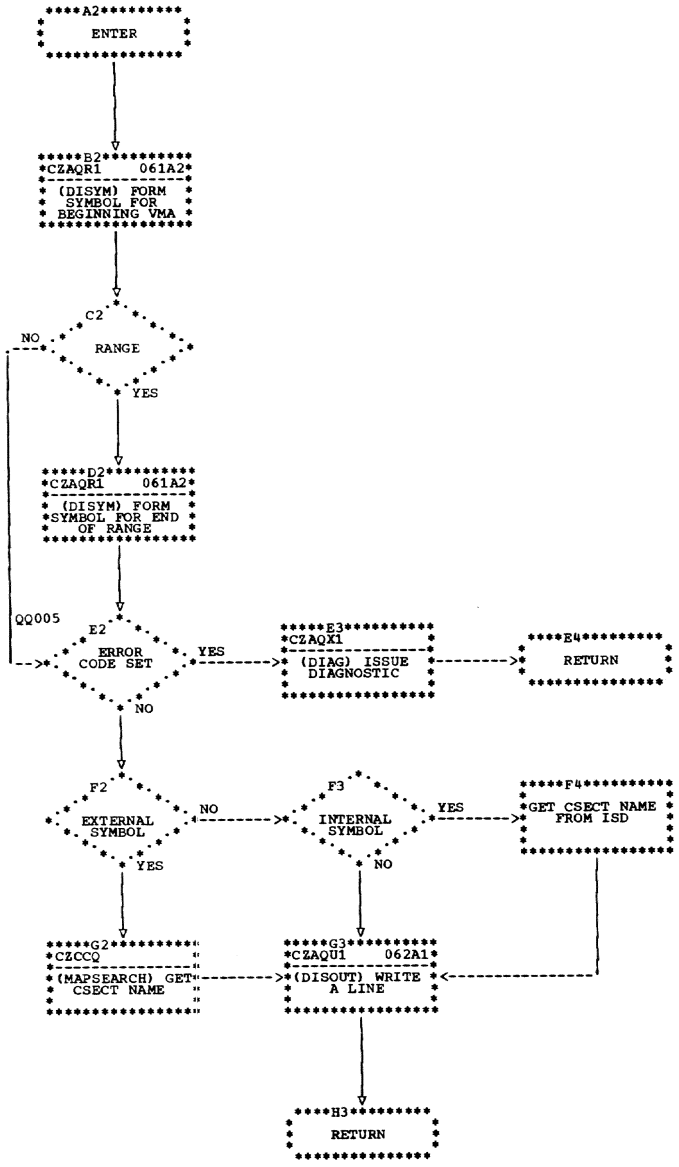
GY28-2014-2

Program Control System

Flowcharts on pages 73-136 were not scanned.

CHART BZ. DISRHEAD (FORMAT RANGE HEADER) - CZAQQ

PAGE 1 OF 1



APPENDIX A: PCS MNEMONIC CROSS-REFERENCE LIST

This appendix lists mnemonics referenced in PCS routines. Only those used extensively or those that have major functions are listed -- simple constants in PSECTs are omitted.

Three types of mnemonics are listed:

1. Subroutine names -- Along with the description, this list gives the entry point label assigned to the subroutine, plus the name of the assembly module CSECT in which the subroutine is located.

2. Data names -- In addition to the description, this list gives the location of each item. Names of items in CSECTs or PSECTs are shown with the control section name as the location. Names of items in a table are shown with the name of the table as the location. The names of the tables themselves show where the storage for the entire table is assigned. The entry point column is not applicable and has been left blank.

3. Mnemonic values -- Mnemonics representing condition status or numeric values of codes are given in this appendix. The entry point and location columns are not applicable and are left blank.

Mnemonic	Entry Point	Location	Description
ACON			Identifies an operand as an address constant
ACTIND		CZAMBP	Action indicator. Set for a DISPLAY, DUMP or SET phrase list entry.
ADCONIND		LOCITEM	Address constant indicator
ADDITEM	CZAQH1	CZAQBC	Convert an item and move to line
ADDRES			Identify an operand as a hexadecimal address
AISD		ISDMAP	Address of ISD
ALEENT		ISDMAP	Address of Linkage Editor entry
ALIGN		CZAMBP	Alignment indicator. If set, check operand alignment and generate code to align if necessary.
ALLOCATE		CZAMBP	Byte length of Polish string entry to be assigned storage.
ALPHA			Identify alphabetic character
ARRAY			Identify an operand as a subscripted array
ASGCODE	NF900	CZAMBC	Routine to allocate storage for generated code.
AT	CZAMF1	CZAMBC	AT phrase routine
ATENTRY			Entry in AT phrase list
ATNEXT		ATENTRY	Address of next STATAB entry
ATVMA		ATENTRY	VMA/LOCTAB entry address

Mnemonic	Entry Point	Location	Description
BACLTB		CZAMBP	Pointer to first LOCTAB page
BACTIONS		CZAMBP	Controls output of standard header for dynamic statement
BASE15		CZAMBP	Contains current generated code cover
BFLAG		CZAMBP	PCS output indicators
BIS			PCS output indicator for immediate statement
BLINE		CZAMBP	Output area for PCS messages/diagnostics
BMCD--			Diagnostic code for determining message text
BRANCH	CZAMB1	CZAMBC	Branch phrase routine
BRUN			PCS output indicator for BRANCH/GO
BSTOP			PCS output indicator for STOP
BUFFER		CZAMBP	Area for collecting characters obtained from source list.
BUFFERX		CZAMBP	Address of next available byte in BUFFER
BYTELNG		CZAMBP	Controls padding/truncating constants
CALL	CZAMG1	CZAMBC	Call phrase routine
CALLIND		CZAMBP	Call phrase indicator
CCON			Character constant identification
CGCPAGE		CZAMBP	Address of current generated code page
CLASS		CZAMAC	Character classification table
CODEGEN	CZANF1	CZAMBC	Code generator control routine
CODEX			Code cover register for PCS
COMCON	CZANH1	CZAMBC	Combine constants routine
COMVAR			Command variable identification
CONAREA		CZAMBP	Conversion area for constants
CONLNG		EXPHEAD	Byte length of longest constant
CONST			Constant identification
CONTINUE			Phrase list continuation identification
COPIND		CURROP	Current operator indicators
COPPRI		CURROP	Current operator priority
COPTYPE		CURROP	Current operator type
CPLPAGE		CZAMBP	Current phrase list page

Mnemonic	Entry Point	Location	Description
CPSPAGE		CZAMBP	Current Polish string page
CSMADD		CZAMBP	Current STATAB/ISDMAP page
CURROP		CZAMBP	Current encoded operator
CURRPHR		CZAMBP	Current phrase list identification
CURRSTAT		TEMPSTAT	Current PCS statement number
CZAMAC		CZAMBC	CSECT for module 1 of PCS
CZAMAP		CZAMBP	PCS PSECT
CZAMA7		CZAMBP	Interrupt control block for generated code
CZAMA8		CZAMBP	PCSOUT data control block
CZAMA9		CZAMBP	Save area for generated code execution
CZAMG2	CZAMG2	CZAMAC	Program call (Phase I)
CZAMIC		CZAMIC	CSECT for module 2 of PCS
CZAQAC		CZAQBC	CSECT for DISPLAY/DUMP module
CZAQAP		CZAQBP	PSECT for DISPLAY/DUMP module
DACTION		DISPLIST	Current phrase identification
DARRAYF		DISPLIST	Array in process indicator
DATAFLD	CZAMI1	CZAMHC	Evaluate data field routine
DATALOC	CZAML1	CZAMHC	Evaluate data location routine
DBEGVML		DISPLIST	VMA of 1st byte of 1st data location
DBEGVML2		DISPLIST	VMA of 1st byte of last data location.
DBR1		DISPLIST	Base register 1 used in formatted instruction indicator.
DBR2		DISPLIST	Base register 2 used in formatted instruction indicator
DBYADJF		DISPLIST	Bypass line adjustment for hex data
DDIAGNO		DISPLIST	Diagnostic output indicator
DDISPMAX		DISPLIST	Line length for SYSOUT line
DDUMPMAX		DISPLIST	Line length for PCSOUT line
DEFTABLE		CZAQBP	Instruction edit and format table
DENDVML		DISPLIST	VMA of last byte of 1st data location
DENDVML2		DISPLIST	VMA of last byte of last data location
DFTIMEF		DISPLIST	First time indicator

Mnemonic	Entry Point	Location	Description
DHEXADDF		DISPLIST	Show locations as hex addresses
DHFLAG		DISPLIST	Header flags
DHFLAG2		DISPLIST	More header flags
DHOLDA		CZAQBP	Output conversion area
DIAGIND		CZAMBP	Diagnostic indicator
DIAGITEM		CZAMBP	Address of character string in error
DIAGNO	CZANW1	CZAMBC	Issue diagnostic routine
DIAG	CZAQX1	CZAQBC	Diagnostic code for determining message text
DIDENT		DISPLIST	1st data location identification
DIDENTTO		DISPLIST	Last data location identification
DIFLAG		DISPLIST	Item flags for 1st data location
DIFLAG2		DISPLIST	Item flags for last data location
DINFORF		DISPLIST	Assembler format indicator
DINHDRF		DISPLIST	Instruction header line indicator
DISALINE	CZAQK1	CZAQBC	Format 1 line of array elements
DISARAY	CZAQJ1	CZAQBC	Format an array
DISDMAP		DISPLIST	Address of ISDMAP entry
DISDPTR		DISPLIST	Address of ISD/PMD/combined dictionary entry
DISHEX	CZAQM1	CZAQBC	Format a hex range
DISHLINE	CZAQN1	CZAQBC	Format 1 line of a hex range
DISINST	CZAQI1	CZAQBC	Formation instruction
DISOUT	CZAQU1	CZAQBC	Output line
DISPDUMP	CZAQA1	CZAQB	DISPLAY/DUMP control routine
DISPLAY	CZAMD1	CZAMBC	DISPLAY phrase routine
DISPLIST		CZAQBP	DISPLAY/DUMP item (DFIT and DSIT)
DISPLNG		CZAQBP	Length of item after output conversion
DISREG	CZAQF1	CZAQBC	Format register
DISRHEAD	CZAQQ1	CZAQBC	Format and write range header and diagnostics
DISYM	CZAQR1	CZAQBC	Format symbol
DITEMS		CZAQBP	Number of items per output line
DLINE		CZAQBP	Output line for DISPLAY/DUMP

Mnemonic	Entry Point	Location	Description
DLINEPTR		DISPLIST	Current line pointer
DLITEMF		DISPLIST	Last item processed indicator
DLNG		DISPLIST	Length of 1st data location
DLNG2		DISPLIST	Length of last data location
DNOBIF		CZAQBP	Number of bytes in field
DNOEQS		CZAQBP	Number of equal items in a range
DNOGRP		CZAQBP	Number of elements in an array group
DNOISDF		DISPLIST	No entry in ISD indicator
DNOLINE		CZAQBP	Number of items per output line
DNOSUB		DISPLIST	Number of subscripts/dimensions
DOCTABLE		CZAQBC	Table of instruction mnemonics
DOFF		DISPLIST	Subscript/offset
DOFFLAG		DISPLIST	Offset in item indicator
DOFFRNG		DISPLIST	Offset range indicator
DOPCHAR		DOPTAB	Operators using two characters
DOPTAB		CZAMBP	Double character operator table
DOUBLE			Double precision register identification
DPAREPTR		DISPLIST	Current phrase list position
DPARLPTR		DISPLIST	End of phrase list
DQISDMAP		DISPLIST	ISDMAP number of implicit qualifiers
DQUALF		DISPLIST	Suppress qualification indicator
DRNGERR		DISPLIST	Range error indicator
DRNGLNG		CZAQBP	Range length
DSECNO		DISPLIST	Control section number
DSTOPF		DISPLIST	Display/Dump Stop indicator
DSUBDIM		DISPLIST	Subscript dimension
DSUBFLD		DISPLIST	Format subfields in overlay indicator
DSUBVAL		DISPLIST	Subscript value
DTEMPLOC		DISPLIST	CSECT displacement
DTYPE		DISPLIST	Data type
DUMP	CZAMD2	CZAMBC	DUMP phrase routine

Mnemonic	Entry Point	Location	Description
DYNAMIND		CZAMBP	Dynamic statement indicator
EMBEDDED		CZAMBP	Embedded blanks indicator
ENTX			Entry register 15
ERROR			Operand in error identification
EVALUATE		CZAMBP	Valid operands for evaluate indicators
EXPARI			Arithmetic expression indicator
EXPDIAG		CZAMBP	Expression diagnostic code storage
EXPHEAD		CZAMBP	Expression header
EXPIND		CZAMBP	Expression unary indicators
EXPLICIT			Explicitly qualified indicator
EXPLOG			Logical expression indicator
EXPLPC		CZAMBP	Expression left parenthesis count
EXPREL			Relational expression indicator
EXPRESS			Expression identification
EXPSCAN	CZAMH1	CZAMHC	Form Polish string for expression
EXPTYPE		EXPHEAD	Operator types in expression
EXTERN			External symbol identification
EXTERNAL	CZAMO1	CZAMHC	Evaluate external symbol
FATAL			Fatal diagnostic indicator
FCON			Floating point constant identification
FGCPAGE		CZAMAP	First generated code page
FINDLOC	CZAPC1	CZAMBC	Final LOCTAB entry
FINDREAL	CZAPL1	CZAMHC	Final original VMA of instruction
FINDVMA		CZAMBP	VMA to be used by FINDLOC
FIRSTOP		CZAMBP	Polish string address for expression header
FLDDELIM		FLDITEM	Data field delimiter
FLDID1		FLDITEM	1st data location identification
FLDID2		FLDITEM	2nd data location identification
FLDILNG		FLDITEM	Field item length for phrase list
FLDITEM		CZAMBP	Data field item
FLDLNG		FLDITEM	Data length of field

Mnemonic	Entry Point	Location	Description
FLDNAME		FLDITEM	Pointer to symbol name
FLDOFF1		FLDITEM	Subscript offset Polish string and generated code
FLDTYPE		FLDITEM	Data type of field
FLDVMA		FLDITEM	VMA of field
FORMDIAG	CZAPI1	CZAMBC	Form PCS output diagnostic
FORMED		CZAMBP	Source list item outstanding
FORMHEAD	CZAPF1	CZAMBC	Form standard output header
FORMPSW	CZAPJ1	CZAMBC	Format VPSW
FORTRAN		CZAMBP	Symbol in FORTRAN ISD indicator
FPCIND		CZAMBP	Floating point constant indicator
FPLPAGE		CZAMBP	First phrase list page
FPRUT		REGUT	Floating point register usage table
FPSPAGE		CZAMBP	First Polish string page
FPUSED		CZAMBP	Floating point registers available
FSMADD		CZAMBP	First STATAB/ISDMAP page
FZ			Floating point register 0
F6			Floating point register 6
GCINTYP		CZAMBP	Generated code interrupt type
GENCALL	CZAPN1	CZAMBC	Execute generated code
GENCODE			Generated code DSECT
GENERAL			General register identification
GETBASE	CZANV1	CZAMBC	Assign base register
GETCHAR	CZAMQ2	CZAMBC	Get next character
GETPAGE	CZANZ1	CZAMBC	Get 1 page of virtual memory
GETREG	CZAOD1	CZAMBC	Assign register to data
GO	CZAMC2	CZAMBC	Go phrase routine
GPRUT		REGUT	General purpose register usage table
GPUSED		CZAMBP	General purpose registers available
HCON			Hex constant identification
HEADADD		CZAMBP	Phrase list header address

Mnemonic	Entry Point	Location	Description
HEADER		CZAMBP	Phrase list header formed
ICON			Integer constant identification
IDENT1		ITEM	Identification of phrase list entry
IDENT2		ITEM	Identification of phrase list entry
IF	CZAME1	CZAMBC	IF phrase routine
IMPLICIT			Implicitly qualified indicator
IMPQUAL		CZAMBP	ISDMAP number for implicit qualifier
IMPQUAL1		CZAMBP	Primary qualifier ISDMAP entry
IMPQUAL2		CZAMBP	Secondary qualifier ISDMAP entry
INTERN			Internal symbol identification
ISDFOUND		CZAMBP	ISDMAP number for explicit qualifier
ISDMAP			DSECT for ISDMAP entry
ITEM			DSECT for phrase list entries
LINENO		CZAQBP	PCSOUT line number
LGCPAGE		CZAMBP	Last generated code page
LINE	CZAPH1	CZAMBC	Output line for PCS output
LOADOP	CZANT1	CZAMHC	Generate code to load operand
LOCDELIM		LOCITEM	Data location delimiter
LOCENTL		LOCITEM	Data location item entry length
LOCID		LOCITEM	Data location identification
LOCILNG		LOCITEM	Data location item length
LOCITEM		CZAMBP	Data location item
LOCLNG		LOCITEM	Data length
LOCNAME		LOCITEM	Pointer to data location name
LOCOFF		LOCITEM	Polish string/generated code for subscript/ offset
LOCTAB			DSECT for LOCTAB entry
LOCTYPE		LOCITEM	Data type
LOCVMA		LOCITEM	Data location VMA
LPLPAGE		CZAMBP	Last phrase list page
LSMADD		CZAMBP	Last STATAB/ISDMAP page

Mnemonic	Entry Point	Location	Description
LTDELTA		LOCTAB	Length of recomposed instruction
LTINUSE			LOCTAB entry in use indicator
LTOPCODE		LOCTAB	Two bytes replaced by SVC
LTRTN			LOCTAB entry for recomposed instruction
LTRTNADD		LOCTAB	VMA of next sequential instruction
LTSTPTR		LOCTAB	Address of lst STATAB entry
LTSVCLOC		LOCTAB	SVC interrupt address
LTTYPE		LOCTAB	LOCTAB entry type
NAME		ISDMAP	Module name
NEST		CZAMBP	Nesting stack for subscripts/offset
NEXTISD	CZAQDl	CZAQBC	Process next ISD entry
NEXTITEM	CZAQCl	CZAQBC	Process display list
NEXTLIST	CZAQBl	CZAQBC	Process next phrase list
NEXTPAGE		PGHEAD	Address of overflow page
NONALIGN			Unaligned operand in expression
NONFP			Operand in expression not loadable in floating point register
NONGP			Operand in expression not loadable in general purpose register
NONST			Operands in expression must be loaded
NULL			Operator/delimiter identification
NUMER			Numeric character identification
OFATAL			Operand fatal diagnostic indicator
OFFLINE			Standard header offline indicator
ONLINE			Standard header online indicator
OPBRIX		OPSTACK	Operator branch index
OPCHAR		OPTAB	Operators using single character
OPERANDI		EXPHEAD	Operand processed indicator
OPGEN	CZANIl	CZAMHC	Operator code generator routine
OPINDEX		CZAMBP	OPSTACK index
OPSTACK			DSECT for operators
OPTAB		CZAMBP	Single character operator table

Mnemonic	Entry Point	Location	Description
OPTYPE-		CZAMBP	Valid operands for evaluation
PCODE-			Codes determining message text and responses
PCSDCB		CZAMBP	Data control block for PCSPUT
PCSDIAGS		CZAMBP	Storage for diagnostic codes
PCSPUT	CZAPB1	CZAMBC	Phase III PCS Output Control routine
PERCNT			Dynamic count identification
PERMBOT		PGHEAD	Permanent storage assigned from bottom of page
PERMPMD		CZAMBP	PMD table index
PERMTOP		PGHEAD	Permanent storage assigned from top of page
PGHEAD			DSECT for page control
PHASE2	CZANA1	CZAMBC	Phase II PCS Input Control
PHRASEID		HEADER	Current phrase identification
PHRASELL		HEADER	Current phrase list length
PHRASELQ		HEADER	Current implicit qualifier
PLDELTA		PLHEAD	Phrase list length
PLHEAD			DSECT for phrase list header
PLIDENT		PLHEAD	Phrase list identification
PLINEPTR		CZAMBP	Line pointer for PCS
PLLOC		CZAMBP	VMA to be converted to a symbol
PLNEXT		PLHEAD	Next word in phrase list
PLPOINT		TEMPSTAT	Phrase list pointer
PLQUAL		PLHEAD	Phrase list qualifier
PLWORD-		HEADER	Single entry phrase list
PMDVMA		CZAMBP	Table of PMDs flagged for unloading
POLARITH			Polish String arithmetic unary indicator
POLBASE			Base assigned for Polish string operand indicator
POLCON			Polish string constant indicator
POLCTRL			Polish string control entry
POLDIM			Polish string dimension entry
POLDIMD		POLISH	Dimension displacement in ISD

Mnemonic	Entry Point	Location	Description
POLDL		POLISH	Data length
POLENTRY		CZAMAP	Polish string entry address
POLHEAD			Polish string header ident
POLIND1		POLISH	Miscellaneous polish string flags
POLIND2		POLISH	More polish string flags
POLINREG			Operand in register flag
POLISH			DSECT for polish string
POLLOG			Polish string logical unary indicator
POLOP		OPSTACK	Encoded operator
POLOPT			Polish string operator flag
POLOVF			Polish string overflow ident
POLPRI		OPSTACK	Operator priority
POLSTART		CZAMBP	Start of generated code
POLSTORE			Polish string stored subexpression flag
POLSUB			Polish string subscript entry
POLTERM			Polish string terminator ident
POLTYPE		EXPHEAD	Expression data type
POLVMA		POLISH	Operand VMA
POWERTAB		CZAMHC	Power of 10 table
PQNDELIM		PQNITEM	Item delimiter
PQNID		PQNITEM	Item identification
PQNITEM		SLITEMS	Primary qualifying name item
PQNPTR		CZAMBP	ISDMAP entry address of primary qualifier
PROMPT	CZANX1	CZAMBC	Solicit user for response
PSIND		CZAMBP	PSECT index for stored subexpressions
PSOPERND		CZAMBP	Address of Polish string operand
PX			PSECT cover register 11
QUALIFY	CZAMR1	CZAMBC	QUALIFY phrase routine
REALCON		CZAQBC	Floating point conversion
REGUT		CZAMBP	Register usage table
REMOVE	CZAMS1	CZAMBC	REMOVE phrase routine

Mnemonic	Entry Point	Location	Description
RESLNG		CZAMBP	Length of expression result
RESLOAD		CZAMBP	Expression result indicators
RTNX			Return register 14
SAVEX			Save area register 13
SAVIX	CZAPK1	CZAMBC	Executes saved instruction
SCANFLD	CZAMQ1	CZAMBC	Scan field to delimiter
SCANIND		CZAMBP	Diagnostic scan indicator
SET	CZAMA1	CZAMBC	SET phrase routine
SIMVAR	CZAQG1	CZAQBC	Format a simple variable
SINGLE			Single precision register ident
SL		CZAMBP	Search list for link edit ISDs
SLCOUNT		SLITEM	Character string length
SLDELIM		SLITEM	Item delimiter
SLIDENT		SLITEM	Item ident
SLITEM		CZAMBP	Source list item
SLITEMS		CZAMBP	Identified source list items
SOFFSET			Offset identification flag
SOPCHAR		SOPTAB	Subscript operator characters
SOPTAB		OPTAB	Subscript operator table
SQNDELIM		SQNITEM	Item delimiter
SQNID		SQNITEM	Item ident
SQNITEM		SLITEMS	Secondary qualifying name item
SQNPTR		CZAMBP	Secondary qualifier ISDMAP entry address
STACK		CZAMBP	Operator/operand stack
STACKIND		OPSTACK	Operator indicators
STATAB			DSECT for statement table entry
STATABAD		CZAMBP	STATAB entry address number ident
STATNO			Statement number ident
STNUM		STATAB	Statement number
STOP	CZAMC1	CZAMBC	STOP phrase routine

Mnemonic	Entry Point	Location	Description
STPCOUNT		STATAB	Dynamic count
STPLIST		STATAB	Phrase list address
SUBDELIM		SUBITEM	Item delimiter
SUBDIAG		CZAMBP	Storage for subscript diagnostic codes
SUBDIM		NEST	Subscript dimension entry
SUBGEN	CZANG1	CZAMHC	Generate code for subscripts/offsets
SUBID		SUBITEM	Item ident
SUBINC		SUBDIM	Dimension displacement in ISD
SUBIND		NEST	Subscript unary indicator
SUBIND2		SUBDIM	Subscript entry indicators
SUBISD		SUBDIM	Address of array entry in ISD
SUBITEM		SLITEMS	Statement number subscript item
SUBLPC		CZAMBP	Left parenthesis count for subscripts
SUBNDM		NEST	Number of dimensions
SUBPOL	CZAMJ1	CZAMHC	Form Polish string for subscript/offset
SUBPST		SUBDIM	Base address of PSECT
SUBSTO		CZAMBP	Stored subexpression storage
SVCTAB		CZAMBP	Table of location dependent SVCs
SYMDELIM		SYMITEM	Item delimiter
SYMGEN	CZAPG1	CZAMHC	Generate symbol for VMA
SYMID		SYMITEM	Item ident
SYMIND		CZAMBP	Symbol indicator
SYMITEM		SLITEMS	Symbol item
SYNONYM		CZAMBP	Synonym indicator
TEMPBOT		PGHEAD	Temporary storage assignment from bottom of page
TEMPPMD		CZAMBP	PMD table index
TEMPSTAT		CZAMAP	Immediate STATAB entry
TEMPTOP		PGHEAD	Temporary storage assignment from top of page
TERMINAT			Phrase list terminator
TRAILING		CZAMBP	Trailing blanks indicator
UNDEFINE			Undefined command variable ident

Mnemonic	Entry Point	Location	Description
UNLOAD	CZAMT1	CZAMB1	UNLOAD phrase routine
VALIDOP		EXPHEAD	Valid operator indicator
VALMOD	CZAOA1	CZAMBC	Evaluate module name
VALSYM	CZAOB1	CZAMHC	Evaluate internal symbol
VARLNG		EXPHEAD	Byte length of longest variable
VARLOAD		EXPHEAD	Variable load indicators
VARTYPE		EXPHEAD	Data type of variables
WARN			Warning diagnostic indicator
WZ			Working register 0
W1			Working register 1
W2			Working register 2
W3			Working register 3
W4			Working register 4
W5			Working register 5
W6			Working register 6
W7			Working register 7
W8			Working register 8
W9			Working register 9
W10			Working register 10

APPENDIX B: PCS ROUTINE/ASSEMBLY MODULE CROSS REFERENCE LIST

Routine I.D.	Assembly Module	Routine Name
CZAMA	CZAMB	SET
CZAMB	CZAMB	BRANCH
CZAMC	CZAMB	STOP & GO
CZAMD	CZAMB	DISPLAY & DUMP
CZAME	CZAMB	IF
CZAMF	CZAMB	AT
CZAMG	CZAMB	CALL
CZAMH	CZAMH	EXPSCAN - Expression Scan
CZAMI	CZAMH	DATAFLD - Form Data Field Definition
CZAMJ	CZAMH	SUBPOL - Subscript to Polish
CZAML	CZAMH	DATALOC - Form Data Location
CZAMO	CZAMH	EXTERNAL - Form External Symbol
CZAMQ	CZAMB	SCANFLD & GETCHAR
CZAMR	CZAMB	QUALIFY
CZAMS	CZAMB	REMOVE
CZAMT	CZAMB	UNLOAD
CZANA	CZAMB	PHASE2 - Phase II PCS Input Control
CZANF	CZAMB	CODEGEN - Code Generator
CZANG	CZAMH	SUBGEN - Subscript Generator
CZANH	CZAMB	COMCON - Combine Constants
CZANI	CZAMH	OPGEN - Operator Code Generator
CZANT	CZAMH	LOADOP - Load Operand
CZANV	CZAMB	GETBASE - Base Register Assignment
CZANW	CZAMB	DIAGNO - Issue Diagnostics
CZANX	CZAMB	PROMPT - User Prompting
*CZANZ	CZAMB	GETPAGE - Allocate Virtual Storage
CZAOA	CZAMB	VALMOD - Evaluate Module Name
CZAOB	CZAMH	VALSYM - Evaluate Symbol
CZAOD	CZAMB	GETREG - Register Assignment
CZAPB	CZAMB	PCSPUT - Phase III PCS Output Control
CZAPC	CZAMB	FINDLOC - Location Table Scan
CZAPG	CZAMH	SYMGEN - Symbol Generator
*CZAPH	CZAMB	LINE
CZAPI	CZAMB	FORMDIAG - Format Diagnostics
CZAPK	CZAMB	SAVIX - Saved Instruction Execution
*CZAPL	CZAMH	FINDREAL - Find Real Address
CZAPN	CZAMB	GENCALL - Call Generated Code
CZAQA	CZAQB	DISPDUMP - Display/Dump Control
CZAQB	CZAQB	NEXTLIST - Process Phrase List
CZAQC	CZAQB	NEXTITEM - Process Display List
CZAQD	CZAQB	NEXTISD - Process Next ISD Entry
CZAQF	CZAQB	DISREG - Display Registers
CZAQG	CZAQB	SIMVAR - Display Simple Variable
CZAQH	CZAQB	ADDITEM - Convert an Item by Data Type
CZAQI	CZAQB	DISINST - Display Instruction
CZAQJ	CZAQB	DISARAY - Display Array
CZAQK	CZAQB	DISALINE - Display a Line of an Array
CZAQM	CZAQB	DISHEX - Display a Range in Hex
CZAQN	CZAQB	DISHLINE - Display a Hex Line
CZAQQ	CZAQB	DISRHEAD - Format Range Header
CZAQR	CZAQB	DISYM - Format Symbol
*CZAQT	CZAQB	DBIN - Output Binary Format
CZAQU	CZAQB	DISOUT - Output a Line
CZAQV	CZAQB	REALCON - Real Number Conversion
*CZAQX	CZAQB	DTAG - Dynamic Diagnostic

*Because of their simplicity, these routines have not been flowcharted.

Routine: PHASE I PCS INPUT, (CZAMA) Level: 1

Routine	Function	Called Routines	Calling Conditions
CZAMA SET	Processes the SET phrase	DATAFLD	Always called.
		EXPSCAN	Always called.
		DIAGNO	If error occurs.
CZAMB BRANCH	Processes the BRANCH phrase	DATAFLD	Always called.
		DIAGNO	If error occurs.
CZAMC STOP/GO	Processes the STOP or GO phrase	DIAGNO	If error occurs.
CZAMD DISPLAY/ DUMP	Processes the DISPLAY or DUMP phrase	DATAFLD	Always called.
		EXPSCAN	If what is to be displayed is the result of an expression.
		DIAGNO	If error occurs.
CZAME IF	Processes the IF phrase	DATAFLD	Always called.
		EXPSCAN	Always called.
		DIAGNO	If error occurs.
CZAMF AT	Processes the AT phrase	DATAFLD	Always called.
		CODEGEN	Only if operand is offset.
		GENCALL	Only if CODEGEN is called.
		DIAGNO	If error occurs.
CZAMG CALL	Processes explicit and implicit CALL phrase	EXTERNAL	Called when processing implicit phrase.
		DATAFLD	Called when processing explicit phrase or if parameters are present.
		EXPSCAN	If parameters are present.
		DIAGNO	If error occurs.
CZAMR QUALIFY	Processes QUALIFY phrase	SCANFLD	Always called.
		VALMOD	Always called.
		DIAGNO	If error occurs.

Routine: PHASE I PCS INPUT, (CZAMA) Level: 1 (cont'd)

Routine	Function	Called Routines	Calling Conditions
CZAMS REMOVE	Processes REMOVE phrase	DATALOC	Always called.
		DIAGNO	If error occurs.
CZAMT UNLOAD	Removes all effects of PCS in user's task	None	

Routine: PHASE I PCS INPUT, (CZAMA) Level: 2

Routine	Function	Called Routines	Calling Conditions
CZAMH EXPSCAN	Forms Polish string in an expression	DATAFLD	If there is a second operand i.e., if expression is multiple term.
		SCANFLD	If an operator is present.
		DIAGNO	If error is present.
		PROMPT	If the expression type is undefined.
CZANF CODEGEN	Generates code.	(See Phase II chart for this routine)	
CZAPN GENCALL	Executes generated code	(See Phase III chart for this routine)	

Routine: PHASE I PCS INPUT, (CZAMA) Level: 3

Routine	Function	Called Routines	Calling Conditions
CZAMI DATAFLD	Forms data field items.	DATALOC	Always called.
		SUBPOL	If there is an offset or subscript.
		DIAGNO	If error occurs.
CZANX PROMPT	Issues message asking for user response	DIAGNO	If nonconversational task.

Routine: PHASE I PCS INPUT, (CZAMA) Level: 4

Routine	Function	Called Routines	Calling Conditions
CZAMJ SUBPOL	Forms Polish string for subscript/offset expression.	DATALOC	Always called.
		SCANFLD	Always called.
		DIAGNO	If error occurs.

Routine: PHASE I PCS INPUT, (CZAMA) Level: 5

Routine	Function	Called Routines	Calling Conditions
CZAML DATALOC	Forms a data location item.	SCANFLD	Always called.
		VALMOD	If there is an explicit qualifier to be evaluated (i.e., a module ISD must be found).
		VALSYM	If an internal symbol is possible.
		EXTERNAL	If an external symbol is possible.
		DIAGNO	If error occurs.
		GETCHAR	If hex address, hex constant, or character constant.

Routine: PHASE I PCS INPUT, (CZAMA) Level: 6

Routine	Function	Called Routines	Calling Conditions
CZAMO EXTERNAL	Forms a data location item for an external symbol, an undefined command variable, or a defined command variable.	None	
CZAMQ1 SCANFLD	Forms a source list item.	GETCHAR	Always called.
CZANW DIAGNO	Issues diagnostics.	GETCHAR	Called for diagnostic scan.
CZAOA VALMOD	Evaluates module name and locates ISD for the module.	None	
CZAOB VALSYM	Forms a data location item for an internal symbol.	None	

Routine: PHASE I PCS INPUT, (CZAMA) Level: 7

Routine	Function	Called Routine	Calling Conditions
CZAMQ2 GETCHAR	Gets next character.	None	

Routine: PHASE II PCS INPUT, (CZANA) Level: 1

Routine	Function	Called Routines	Calling Conditions
CZANA PHASE2	Generates code and assigns permanent storage for final phrase list processing.	CODEGEN	If there is any code to be generated.
		FINDLOC	If statement is dynamic.
		PCSPUT	If the statement is immediate.
		DIAGNO	If fatal error occurs.
		PROMPT	If non-fatal error occurs.

Routine: PHASE II PCS INPUT, (CZANA) Level: 2

Routine	Function	Called Routines	Calling Conditions
CZANF CODEGEN	Controls the execution of generated code; it also generates the final return linkage from generated code to PCS.	OPGEN	If one of the operands is not a constant.
		COMCON	If both operands are constants.
		SUBGEN	If subscript/offset operators are present.
		GETBASE	If base register for result has not been assigned.
		LOADOP	If result has not been loaded but must be loaded.
CZANX PROMPT	Issues diagnostic and solicits user response.	DIAGNO	If nonconversational task
CZAPB PCSPUT	Performs immediate actions.	(See Phase III chart for this routine)	
CZAPC FINDLOC	Locates a LOCTAB entry.	None	

Routine: PHASE II PCS INPUT, (CZANA) Level: 3

Routine	Function	Called Routine	Calling Conditions
CZANG SUBGEN	Computes a dimension and performs dimension check and applies subscript/ offset to the base of the symbol.	LOADOP	If subscript/offset is variable.
		GETBASE	If subscript/offset is variable.
		DIAGNO	If dimension check on constant subscript has failed.
CZANH COMCON	Performs specified operations on two constants.	DIAGNO	If program interrupt occurs during processing.
CZANI OPGEN	Generates code to combine two operands by operator specified.	LOADOP	Always called.
		GETREG	If relational operator is being processed.
		GETBASE	If relational operator is being processed.

Routine: PHASE II PCS INPUT, (CZANA) Level: 4

Routine	Function	Called Routine	Calling Conditions
CZANT LOADOP	Generates code to load an operand.	GETBASE	Always called.
		GETREG	Always called.
CZANW DIAGNO	Outputs diagnostic.	None	

Routine: PHASE II PCS INPUT, (CZANA) Level: 5

Routine	Function	Called Routine	Calling Conditions
CZANV GETBASE	Assigns base register for operand; provides alignment if necessary.	GETREG	If operand is a dummy variable.

Routine: PHASE II PCS INPUT, (CZANA) Level: 6

Routine	Function	Called Routine	Calling Conditions
CZAOD GETREG	Assigns data register to contain operand.	None	

Routine: PHASE III PCS OUTPUT CONTROL (CZAPB) Level: 1

Routine	Function	Called Routines	Calling Conditions
CZAPB PCSPUT	Exercises overall control for processing dynamic or immediate PCS entries.	FINDLOC	Always called (dynamically only).
		LINE	If line is to be output.
		DISPDUMP	If a display/dump or set phrase is being processed.
		GENCALL	If there is generated code to be executed.
		FINDREAL	If a GO phrase is being processed.
		SYMGEN	If a BRANCH or GO phrase is being processed.
		SAVIX	Always called (dynamically only).
		FORMDIAG	If there is an illegal entry into PCS.

Routine: PHASE III OUTPUT CONTROL (CZAPB) Level: 2

Routine	Function	Called Routines	Calling Conditions
CZAPG SYMGEN	Translates a VMA into user's symbol (plus offset, if needed)	None	
CZAPI FORMDIAG	Writes a diagnostic message	None	
CZAPK SAVIX	Recomposes overlaid instructions	FINDLOC	Always called.
		LINE	If instruction cannot be recomposed (due to storage limitation)
CZAPL FINDREAL	Finds the virtual memory address for recomposed instruction	FINDLOC	Always called.
CZAQA DISPDUMP	Processes a display/dump, or set list	DISOUT	If qualification has changed.
		NEXTLIST	Always called.

Routine: PHASE III PCS OUTPUT CONTROL (CZAPB) Level: 3

Routine	Function	Called Routines	Calling Conditions
CZAPH LINE	Outputs a line.	None	
CZAQB NEXTLIST	Processes a phrase list entry.	GENCALL	If there is generated code to be executed.
		DISREG	If a list entry for a register is processed.
		DISHLINE	If diagnostic code set.
		NEXTITEM	If non-register list entry is processed.
		NEXTISD	If internal symbol entry is processed.
		DISRHEAD	If a range entry or diagnostic is to be written.

Routine: PHASE III PCS OUTPUT CONTROL (CZAPB) Level: 4

Routine	Function	Called Routines	Calling Conditions
CZAQB	Processes the two display list items (called DISPLIST).	DISARAY	If one DISPLIST item is an array.
		SIMVAR	If a simple variable DISPLIST item is processed.
		DISHEX	If a DISPLIST item for hexadecimal addresses is processed.
		NEXTISD	If a DISPLIST item for an internal symbol range is processed.
		DISYM	If a range is processed.
		DISOUT	If a line is to be output.
		DBIN	If binary is to be output.
CZAQF DISREG	Processes a phrase list entry for a register.	GENCALL	If a set phrase is processed.
		DISOUT	Always called.
		ADDITEM	Always called.
CZAQQ DISRHEAD	Formats and writes range headers and diagnostics.	DISYM	Always called.
		DISOUT	If a range header is to be printed.
		DIAG	If diagnostic condition detected.

Routine: PHASE III PCS OUTPUT CONTROL (CZAPB) Level: 5

Routine	Function	Called Routines	Calling Conditions
CZAPN GENCALL	Executes generated code.	FORMDIAG	If a program interrupt occurred in generated code, or if generated code returned an error code.
CZAQD NEXTISD	Formats a DISPLIST item.	None	
CZAQJ DISARAY	Formats an array.	DISYM	If a range is not processed.
		DISOUT	If a line is full.
		DISALINE	Always called.
		ADDITEM	Always called.
CZAQM DISHEX	Formats a hexadecimal range.	DISHLINE	Always called.
		DISOUT	If there is equal line suppression.
		SIMVAR	If an instruction is being processed.
DBIN CZAQT	Formats a line of binary.	DISYM	If a range is not processed
		DISOUT	Always called.
CZAQX DIAG	Writes diagnostic messages.	None	

Routine: PHASE III PCS OUTPUT CONTROL (CZAPB) Level: 6

Routine	Function	Called Routines	Calling Conditions
CZAQG SIMVAR	Formats a simple variable.	DISYM	Always called.
		ADDITEM	Always called.
		DISOUT	Always called.
		DISINST	If an instruction is being processed.
CZAQK DISALINE	Formats an array line.	ADDITEM	Always called.
		DISOUT	Always called.
CZAQN DISHLINE	Formats a hexadecimal data line.	DISOUT	Always called.

Routine: PHASE III PCS OUTPUT CONTROL (CZAPB) Level: 7

Routine	Function	Called Routines	Calling Conditions
CZAQH ADDITEM	Converts an item and adds it to the line.	DISOUT	If line is full.
		REALCON	If floating-point numbers are to be converted.
CZAQI DISINST	Formats an instruction	FINDLOC	If the instruction is a PCSVC.
		DISOUT	Always called.
CZAQR DISYM	Forms a symbol	None	

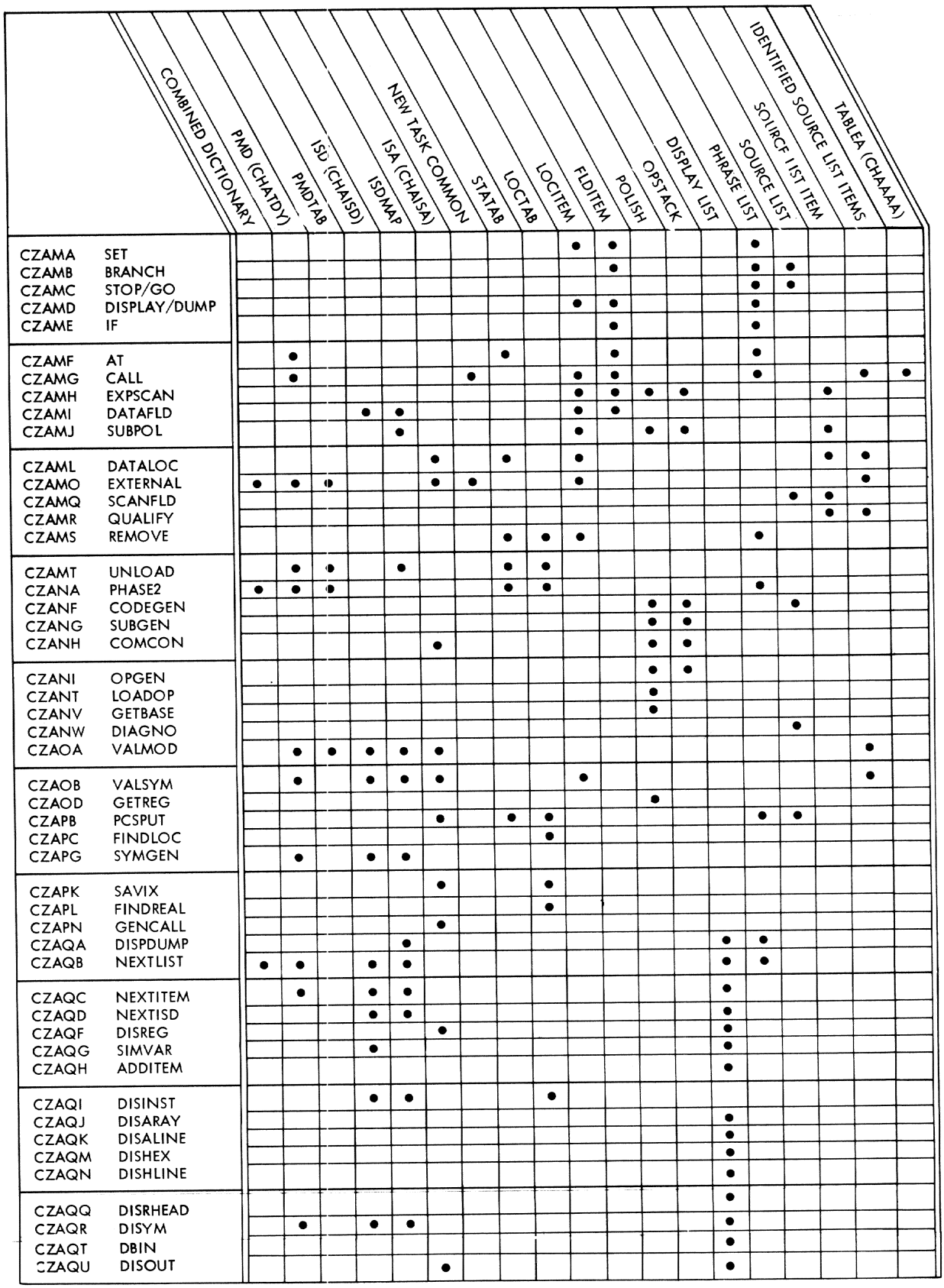
Routine: PHASE III PCS OUTPUT CONTROL (CZAPB) Level: 8

Routine	Function	Called Routines	Calling Conditions
CZAPC FINDLOC	Finds a LOCTAB entry.	None	
CZAQU DISOUT	Writes a line.	None	
CZAQV REALCON	Converts and formats a floating-point number.	None	

APPENDIX D: PCS LINKAGES TO EXTERNAL ROUTINES

If a routine has no external linkage, the PCS subroutine name has been omitted from the table.

PCS ROUTINE		EXTERNAL ROUTINE																						
		BY MACRO															BY TYPE I CALL						BY SVC
		CKCLS	PIREC	DLINK	SIR	DIR	INTINQ	GETMAIN	FREEMAIN	VISAM OPEN	VISAM PUT	FIND	GATWR	PRMPT	GDV	EBCDTIME	MAPSEARCH (CZCCQ)	HASHSEARCH (CZCCF)	DICTIONARY HANDLER (CZASD)	SOURCE LIST HANDLER (CZASC)	MOVEPAGE (CZCOC)	INTERVENE (CZAMZ)	DYNAMIC LOADER (CZCDL1)
CZAMB	BRANCH		X																					
CZAMC	STOP/GO		X																					
CZAMF	AT	X							X								X							
CZAMI	DATAFLD	X																						
CZAMJ	SUBPOL	X																						
CZAMO	EXTERNAL			X														X	X					X
CZAMQ	SCANFLD																		X	X				
CZAMT	UNLOAD								X															
CZANA	PHASE2								X	X			X						X					
CZANF	CODEGEN																			X				
CZANH	COMCON				X	X	X																	
CZANW	DIAGNO												X	X	X									
CZANX	PROMPT													X										
CZANZ	GETPAGE							X																
CZAOA	VALMOD							X				X						X			X		X	
CZAOB	VALSYM																	X						
CZAPB	PCSPUT									X										X		X		
CZAPG	SYMGEN																X							
CZAPH	LINE												X											
CZAPI	FORMDIAG													X										
CZAPN	GENCALL				X	X	X																	
CZAQA	DISPDUMP														X									
CZAQB	NEXTLIST	X																						
CZAQC	NEXTITEM															X								
CZAQU	DISOUT										X		X											


COMBINED DICTIONARY
PMD (CHATDY)
PMDTAB
ISD (CHAISD)
ISDMAP
NEW TASK COMMON
ISA (CHAISA)
STATAB
LOCTAB
LOCITEM
FLDITEM
POLISH
OPSTACK
DISPLAY LIST
PHRASE LIST
SOURCE LIST
IDENTIFIED SOURCE LIST ITEM
TABLEA (CHAAAA)
CZAMA SET
CZAMB BRANCH
CZAMC STOP/GO
CZAMD DISPLAY/DUMP
CZAME IF
CZAMF AT
CZAMG CALL
CZAMH EXPSCAN
CZAMI DATAFLD
CZAMJ SUBPOL
CZAML DATALOC
CZAMO EXTERNAL
CZAMQ SCANFLD
CZAMR QUALIFY
CZAMS REMOVE
CZAMT UNLOAD
CZANA PHASE2
CZANF CODEGEN
CZANG SUBGEN
CZANH COMCON
CZANI OPGEN
CZANT LOADOP
CZANV GETBASE
CZANW DIAGNO
CZAOA VALMOD
CZAOB VALSYM
CZAOD GETREG
CZAPB PCSPUT
CZAPC FINDLOC
CZAPG SYMGEN
CZAPK SAVIX
CZAPL FINDREAL
CZAPN GENCALL
CZAQA DISPDUMP
CZAQB NEXTLIST
CZAQC NEXTITEM
CZAQD NEXTISD
CZAQF DISREG
CZAQG SIMVAR
CZAQH ADDITEM
CZAQI DISINST
CZAQJ DISARAY
CZAQK DISALINE
CZAQM DISHEX
CZAQN DISHLINE
CZAQQ DISRHEAD
CZAQR DISYM
CZAQT DBIN
CZAQU DISOUT

APPENDIX F: INTERNAL AND EXTERNAL TABLE REFERENCE DATA

This appendix contains the following external and internal data used by PCS: common areas, PCS communication areas, and PCS internal lookup tables.

COMMON AREAS

PCS common areas consist of: task dictionary, combined dictionary, internal symbol dictionaries (ISD), new task common (NTC), interrupt storage area (ISA), and the source list.

TASK DICTIONARY TABLE (TDY)

The task dictionary table (TDY) (Figures 17, 18, 19, 20, 21) contains information needed to load (and unload) the modules in a particular task. It consists of a heading, two hash tables (system and user), the storage map table (MAP), and one program module dictionary (PMD) for each module loaded during the task. PMDs are arranged in irregularly-located PMD groups. The TDY is initialized by STARTUP and maintained by the dynamic loader. PCS uses the information in the TDY to resolve symbolic references in a PCS statement.

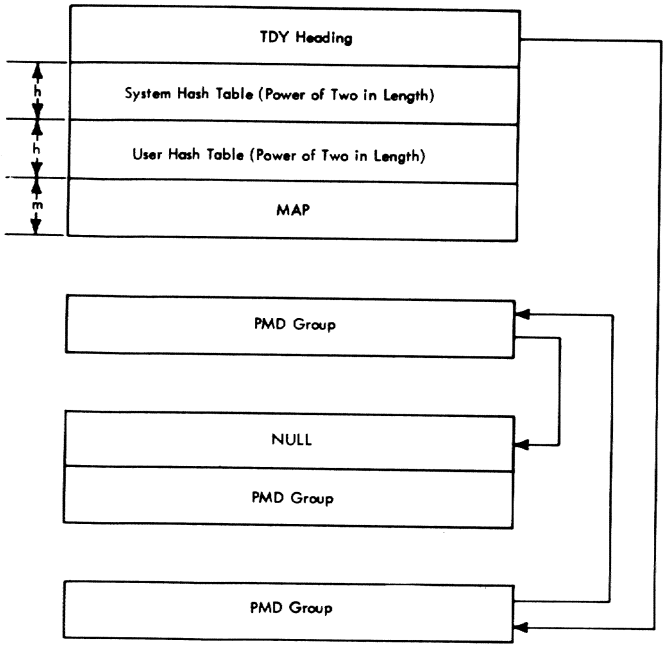


Figure 16. Task Dictionary Organization

TDY Heading (CHATDH)

The TDY heading (Figure 18) is 16 words in length and contains the following:

Word 0 - Link to PMD group
The address of the first word of the last PMD group to be entered into the TDY.

Word 1 - Hash divisor
This is some number not more than the length of each hash table in words. It is provided by STARTUP and remains unchanged during a task.

Word 2 - Pointer to system hash table
The virtual storage address of the beginning of the system hash table.

Word 3 - Pointer to user hash table
The virtual storage address of the beginning of the user hash table.

Note: If the user authority is P or 0, LOGON sets word 3 (pointer to user hash table) equal to the contents of word 2 (pointer to system hash table).

Word 4 - The virtual storage address of the origin of MAP table.

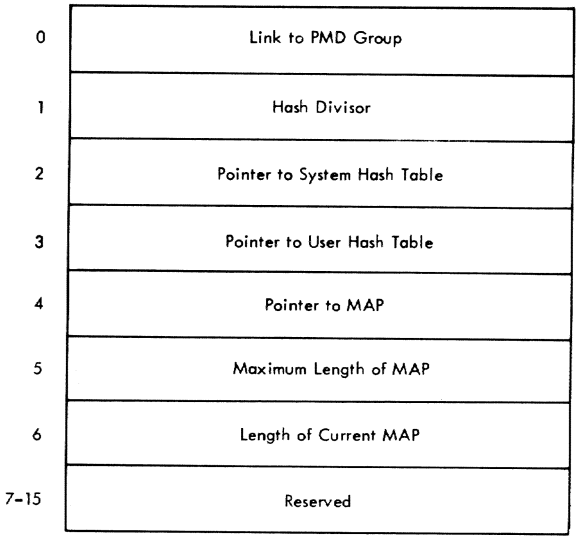


Figure 17. TDY Heading

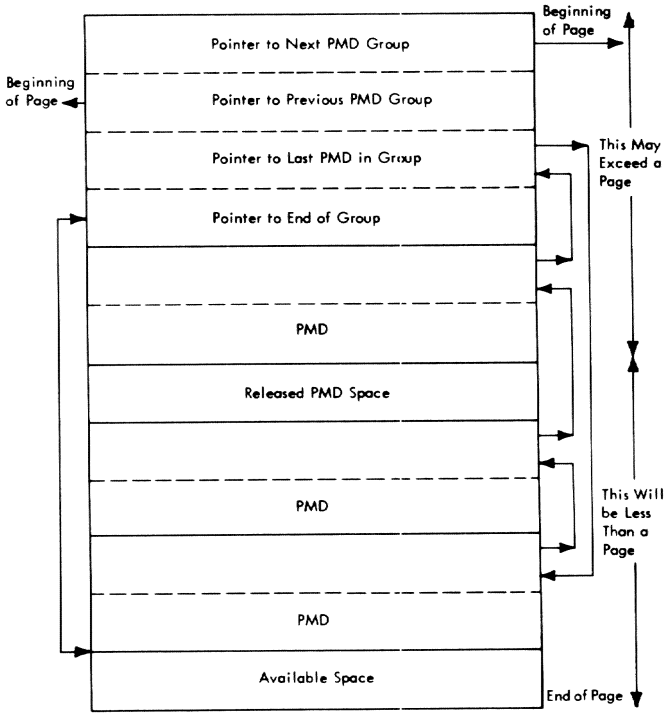


Figure 18. Sample PMD Group

Word 5 - The length, in bytes, of the maximum space allocated by the system for the task's storage MAP.

Word 6 - A count of currently valid MAP entries.

Words 7-15- Reserved for future expansion

Program Module Dictionary (PMD) Group

Each PMD group consists of at least one PMD with its associatd PMD preface (Figure 19). When a new PMD is to be added to the TDY by the dynamic loader, if room exists in the same page that contains the last inserted PMD, the new PMD will be added to that page and become part of that PMD group. If such space does not exist on the page, a new PMD group is formed, starting with the new PMD. Note that the first PMD in a group may exceed a page in length, but that successive PMDs in a group may not exceed a page.

PMD groups (Figure 18) are chained together bi-directionally through the first two words in each PMD group header. The TDY heading contains a pointer to the beginning of the chain of PMD groups.

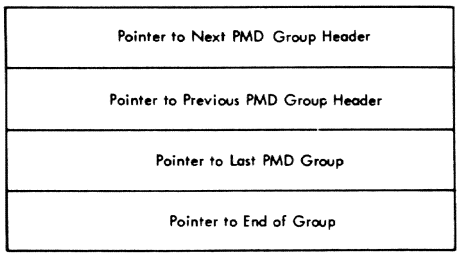


Figure 19. PMD Group Header

PMD Group Header

Each PMD group header (Figure 20) consists of four pointer:

Word 0 - A pointer to the next PMD group header.

Word 1 - A back pointer to the previous PMD group header.

Word 2 - A pointer to the last PMD in this group.

Word 3 - A pointer to the first byte past the end of this PMD group, which therefore defines the beginning of available space in this group.

The PMD group header is at the beginning of a page.

Pointer to Next PMD Group Header: This either contains the address of the next PMD group header, or is zero if this is the last PMD group in the chain.

Pointer to Previous PMD Group Header: This contains the address of word 0 of the previous PMD group header in the chain. The most recently added group will backlink to word 0 of the TDY heading (CHATDH).

Pointer to Last PMD in This Group: This is the beginning of a circular chain of all PMDs in the group. It contains the address of the last (most recent) PMD preface in this PMD group.

Pointer to End of Group: This contains the address of the beginning of the available space at the end of the last page in the group. Space made available by deletion of a PMD within a group is not accounted for. Space is only released on a full group basis.

PMD Preface

The PMD preface (Figure 21) is generated by STARTUP of the PMD preface is part of initial virtual storage, or by the loader if it is not. The contents are maintained by the loader. It always immediately precedes the PMD at load time, and when the term PMD is used in the following description, the PMD preface is generally meant. The PMD preface contains the following entries:

Word 0 - A link to next PMD in the chain of PMDs within this PMD group. The PMD group header contains a pointer to the last PMD of this group. Since each new PMD is inserted at the beginning of the chain, PMDs are in reverse order of appearance within a PMD group. The link contains the address of the first word of the PMD preface of the next PMD. The last PMD in the chain points to the third word in the PMD group header.

Word 1 - A link to the module usage table entry (MUTE) chain for modules that explicitly call this module.

Word 2 - A link to the MUTE chain for modules that are explicitly called by this module.

Word 3 left-half - The number of explicit links (CALLs/LOADs) to this module. For each explicit link to this module the value of this field is incremented by one.

Word 3 right-half - PMD flags. The PMD flags field is a halfword containing flags used by the loader. The following flags are defined (bits numbered from left to right starting with 0):

Bit 15 - Public flag - this bit is set if this module contains any public CSECTs.

Bit 14 - Candidate flag - this bit is set if this module is on the deletion candidate list.

Word 4 - A pointer to the JFCB for library containing this module.

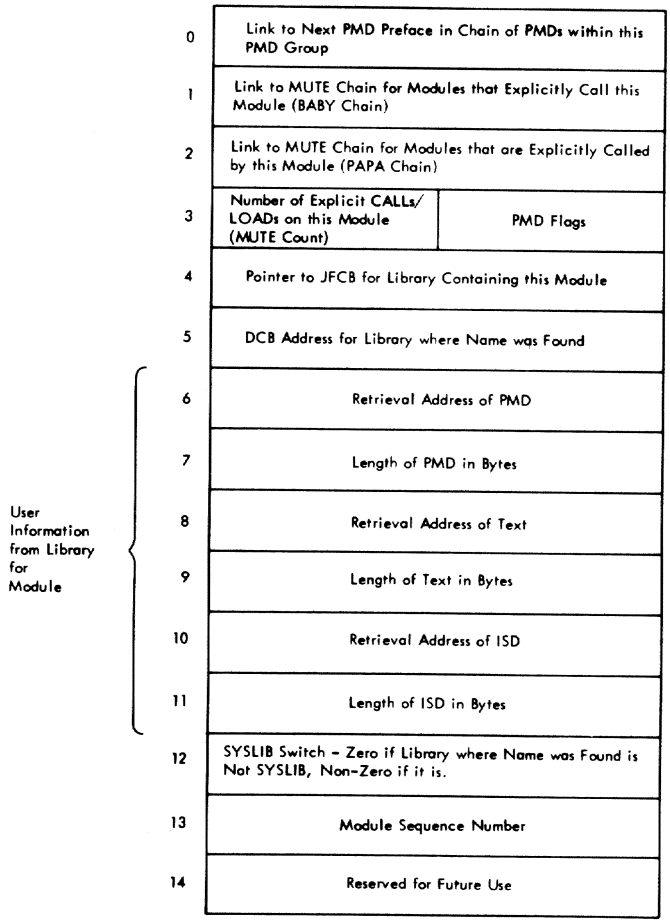
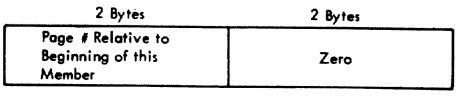


Figure 20. PMD Preface

Word 5 - Address of DCB for library in which name is resolved.

Words 6-11 - User information from library where this module was obtained. The form of the retrieval address is:



The retrieval address for the PMD will, therefore, always be zero.

Word 12 - SYSLIB switch: Set to non-zero if module was extracted from SYSLIB.

Word 13 - The module sequence number. Each module is assigned a consecutive sequence number as it is loaded. This sequence number is used to differentiate

unnamed (non-COMMON) control section references among modules.

Word 14 - Reserved for future use.

Program Module Dictionary (PMD)

The output from an assembler, compiler, or the linkage editor is known as a program module. This is composed of a program module dictionary (PMD), text, and internal symbol dictionary (ISD).

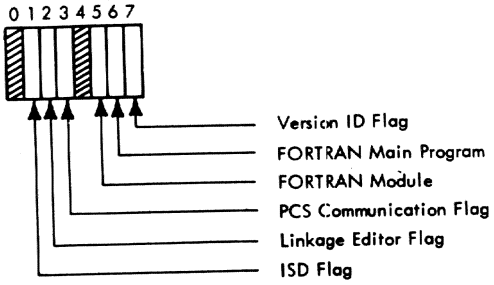
Each PMD consists of one PMD heading plus as many control section dictionaries (CSD) as there are control sections in the module. Address pointers in the PMD are initially relative to the beginning of the PMD itself (not the PMD preface), except where otherwise specified. Some fields in the PMD are filled in by the loader. These are left zero by the language processor. The PMD format is shown in Figure 22.

PMD Heading:

1. Length of PMD in bytes - This length does not include the PMD preface.

2. Diagnostic code (1 byte) - The diagnostic code indicates the highest level diagnostic encountered during generation of the module by the language processor that created it.

3. Flags (1 byte) - The flag bits are numbered from left to right, starting with zero and are defined as follows:



Bit 1
module has an ISD associated. This bit is set by the processor creating the PMD.

Bit 2
module was processed by link editing. This bit is set by the Linkage Editor.

Bit 3
PCS is to be called before module is dynamically unlinked. This bit is set by PCS.

Bit 5
module was produced by the FORTRAN compiler.

Bit 6
FORTRAN module is a main program, not a SUBROUTINE, FUNCTION, or BLOCK DATA subprogram.

Bit 7
version ID indicator. If this bit is set, the module version ID is to be interpreted as a 64-bit binary number which is the creation date of the module. If this bit is not set, the version ID is eight alphameric EBCDIC characters.

4. Length of PMD heading - The length in bytes of the PMD heading.

5. 4-Character I.D. name - The I.D. name is supplied by the user to serve as deck identification if the module is punched into cards. This field is currently unused.

6. Version I.D. - See item 3 (bit 7 discussion) for interpretation of version I.D.

7. Number of REFs for the standard entry point - the DEF for the standard entry point is always treated as a complex DEF. This field contains the number of REFs; it may be zero.

8. Number of modifiers for the standard entry point - This field contains the number of modifiers that are to be used to compute the DEF for the standard entry point.

9. DEF for standard entry point - This seven-word entry describes the DEF for the standard entry point of the module. It has the same form as the individual DEF entries within the CSDs. The standard entry point DEF for the module is considered to belong to the first PSECT of the module and is treated the same as a complex DEF whose ENTRY statement appears within that PSECT.

If no PSECT is declared, the standard entry point will be associated with the first CSECT, instead. This DEF entry contains the following subfields, which are described in the discussion of DEF entries, under control section dictionary.

a. Alphameric name of module
b. Value of DEF
c. R-value displacement
d. CSD link
e. Reserved for future use
f. Search link

The alphameric name is also the name of the module.

10. REF(s) for entry point - Have the same form and function as the REFs described in the CSD discussion that follows.

11. Modifier(s) for standard entry point - These have the same form and function as the modifiers described in the CSD discussion, except that they apply to the standard entry point DEF.

Control Section Dictionary (CSD)

The control section dictionary has the following components:

- CSD heading
- Definition table
- Reference table
- Relocation dictionaries (RLDs)
- Virtual Storage Page Table (VMPT)

CSD Heading:

1. Number of bytes in CSD - Specifies the length of the control section dictionary in bytes.

2. Length of control section in bytes- Specifies the virtual storage span of the control section. The length of the virtual storage page table is derived from this length. For example, if the length of the control section is 8192, the virtual storage page table will contain two pages; but if the length is 8193 bytes, the virtual storage page table will contain three pages. This value will be equal to the highest location counter value assigned by the language processor, plus one.

3. Page number in text of page 0 of CSECT text - The text for each control section in the module occupies an integral number of pages in its resident data set. The text pages for all control sections in a module are contiguous. This number is the page number, relative to the first page of text for this module, of the first page of text for this CSECT. (Numbering begins with 0.)

4. Version I.D. - This is a 64-bit binary number which is the creation date of the control section expressed as the number of microseconds that have elapsed from March 1, 1900 until the time of CSECT creation.

5. PMD link - The PMD link is filled in by STARTUP or the dynamic loader. It points to the beginning of the PMD preface.

6. Whether CXD REF exists and number of QREFs. Bits from left to right contain:

   Bit 0 - set to 0 if no CXD REF exists; set to 1 if a CXD REF does exist. (Only one CXD REF is possible.)

   Bit 1 - not used.

   Bits 2-14 - number of QREFs (contains all zeros if none).

7. Number of implicit references to this control section (user count) - This is a count of the number of REF entries that refer to this control section and are linked to this CSD through their CSD link. It is computed by the loader, and includes both external and internal references. This number is arbitrarily set by STARTUP for each CSECT in initial virtual storage to X'7FFF' to prevent unloading of IVM modules.

8. Number of relocatable definitions - This is the number of relocatable definitions in the definition table. It is always at least one, namely, the control section name DEF.

9. Number of absolute definitions - This is the number of absolute definitions in the definition table. It may be zero.

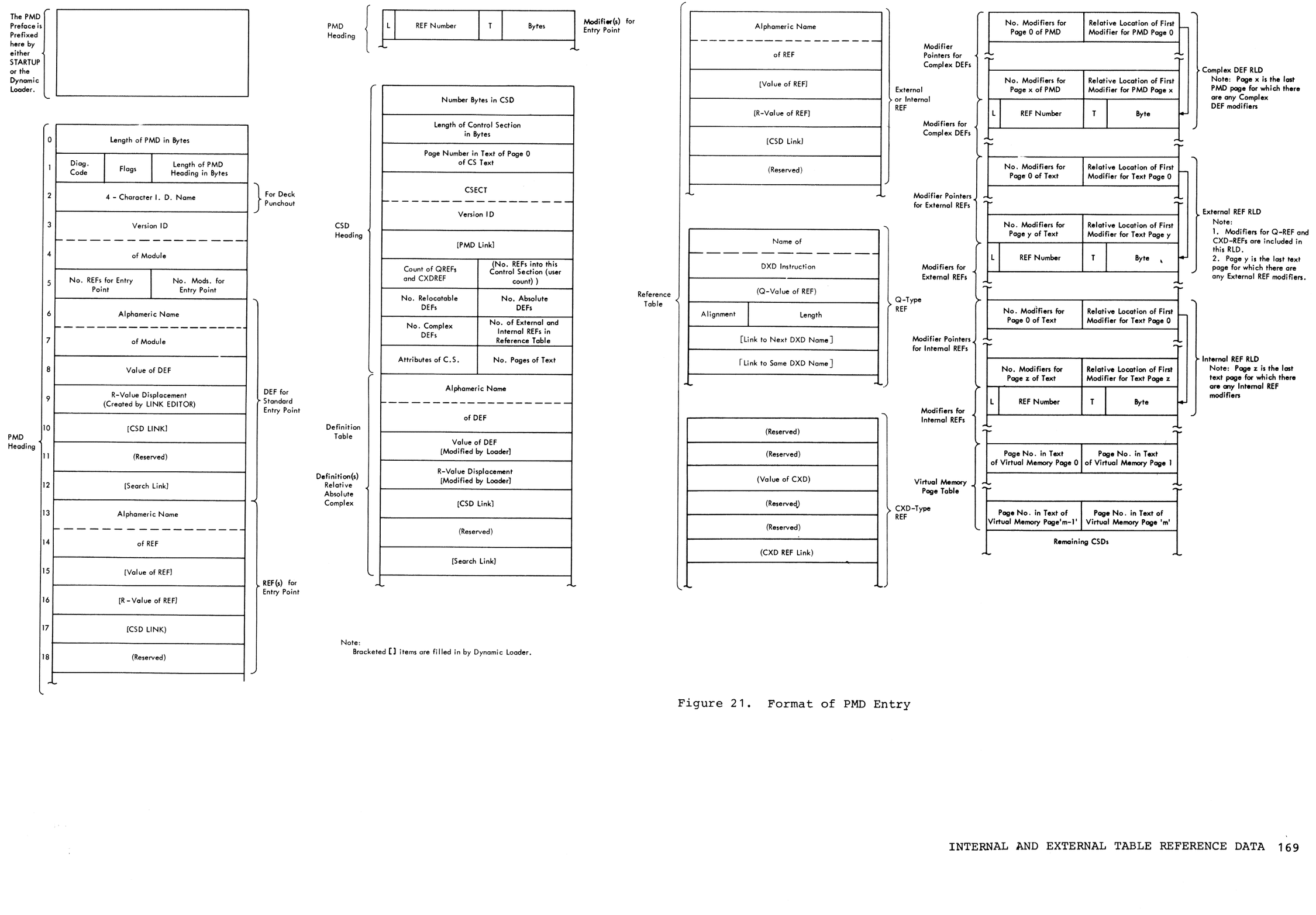


Figure 21. Format of PMD Entry

10. Number of complex definitions - This is the number of complex definitions in the definition table. It may be zero.

11. Number of references from this CSD - This is the sum of external and internal references in the reference table. It may be zero.

12. Attributes - This halfword has one bit set for each attribute possessed by the control section. Currently defined attributes are shown below. Bits are numbered from left to right, starting with 0.

    a. Fixed-length (Bit 14 off) - A fixed-length control section will be allocated a fixed number of pages at load time.

    b. Variable-length (Bit 14 on) - A variable-length control section will be allocated pages in excess of the length stated in the CSD heading.

    c. Read-only (Bit 13 on) - Read-only specifies that the control section may not be stored into. It causes storage protection by means of a storage class-B assignment to all pages of the control section. Non-read-only and nonprivileged control sections are assigned storage class A.

    d. Public (Bit 12 on) - Control sections are not shared by control section name alone. A PUBLIC control section of a module residing in a given data set (library) is shared if another user has access to the same data set and module. Control sections of a given module need not all be PUBLIC or non-PUBLIC. Fixed length PUBLIC control sections with the same attributes are assigned storage in the same assignment. A public control section must never contain relocatable adcons (A-, V-, or R-type).

    e. PSECT (Bit 11 on) - If this bit is set, it causes the dynamic loader to override the system packing indicator and insert this control section as packed.

    f. COMMON (Bit 10) - A COMMON section is a control section common to all modules in which it is declared. COMMON sections are more fully discussed in Linkage Editor and Assembler Language. COMMON sections are of two types:

       (1) Named COMMON sections (those with a name not all blanks). These are treated as fixed-length sections.

       (2) Blank COMMON sections, whose name consists of eight blanks. FORTRAN blank COMMON is assigned the VARIABLE and COMMON attributes by the FORTRAN compiler.

       The treatment of blank COMMON sections differs from that of blank non-COMMON sections. Control section rejection is instituted between blank COMMON sections of different modules, whereas blank non-COMMON sections of different modules are treated as independent control sections. The latter are called unnamed control sections.

    g. Privileged (Bit 9 on) - A control section with a privileged attribute is assigned storage key C, which provides fetch, as well as store, protect. This attribute overrides R/O.

       Anything in a privileged CSECT may be referenced only when the PSW key is zero.

    h. SYSTEM (Bit 8 on) - Any external symbol that appears in a control section which has the SYSTEM attribute cannot be referenced by a user program unless the symbol begins with SYS. Conversely, no references from a control section with a system attribute may be to a user symbol.

    i. TDYCQR validity (Bit 7 on) - The dynamic loader sets this flag to indicate that the count of Q-type REFs in TDYCQR is valid. If bit 7 is off, the count of Q-type REFs is not valid.

    j. Common CSECT Rejected (Bit 6 on) - The dynamic loader sets this flag to indicate to the Program Control System that the CSECT was rejected as a common CSECT that was already loaded in another module.

    k. Bits 4 and 5 are not used.

l. Public Storage Assigned by CONNECT (Bit 3 on) - Set by the dynamic loader, if applicable.

m. PCSA called for this CSD (Bit 2 on) - Set by the dynamic loader, if applicable.

n. CSD has been allocated storage (Bit 1 on) - Set by the dynamic loader, if applicable.

o. Public name (Bit 0 on) - This is used only by the dynamic loader to specify nonblank control sections whose names appear in the shared data set table (SDST). The first such control will appear in the SDST under the module name. A control section may be indicated as both having a public name and rejected.

13. Number of pages of text - This specifies the number of pages of text for this control section in the data set. It should be noted that this generally does not correspond to the number of pages in the virtual storage page table. It cannot, of course, be larger.

Definition Table

The definition table is made up of seven-word entries, one for each external definition in the current control section. Definitions are grouped as relocatable, absolute, and complex in that order. The first definition in the table is the name of the current control section.

Relocatable definitions are external definitions whose value may be computed as the sum of the origin of the control section wherein they appear and a constant that is the symbol's displacement from the section origin.

An absolute definition is an EQU item with an absolute value whose name has been declared an entry point in the control section in which the name is defined.

A complex definition is either an EQU item with a complex relocatable value, i.e., containing external symbols; or a simple relocatable definition whose ENTRY statement appeared within a control section other than the section in which it is defined. The definition entry appears within the CSD of the control section that contains the ENTRY statement. (Note that the origin of the same control section is the R-value for the DEF.) The complex DEF is required in this case, with one REF entry that names the control section in which the DEF symbol is actually defined.

Each DEF in the definition table contains the entries of the following form:

1. Alphameric name of DEF - This field contains the eight-character alphameric name of the DEF.

2. Value of DEF - The value of the DEF is set by the language processor and is modified by STARTUP or the loader, in the case of complex and relocatable definitions. For relocatable DEFs, the value portion of the definition entry contains the displacement value of the symbol relative to the base of its control section. For absolute DEFs, this entry contains the absolute value; for complex DEFs, it contains the absolute portion of the DEF value, which may be zero.

3. R-value displacement - The "displacement for R-value" word contains the displacement of the original defining control section origin with respect to the head of the control section within which the definition now appears. This is required to compute valid R-values for control sections that have been COMBINED by linkage editing. In creating the PMD, only the linkage editor will ever produce a nonzero value in this word.

4. CSD link - This CSD link is initially zero. When the control section is loaded, it is filled in by STARTUP or the dynamic loader as a pointer to the beginning of the CSD in which this DEF appears, providing neither the DEF nor the control section has been rejected.

5. For future use.

6. Search link - This field is filled by the HASH SEARCH routine of either the loader or STARTUP. It contains the address of the beginning of the next DEF entry which hashes to the same value. It contains zero if there are no more DEFs with the same hash value in this chain.

Reference Table, Relocation Dictionaries, and Virtual Storage Page Tables: The reference table, the relocation dictionaries and the virtual storage page table follow the definition table in the CSD. For a complete description of this section of the PMD, see Dynamic Loader PLM.

INTERNAL SYMBOL DICTIONARIES (ISD)

PCS uses the internal symbol dictionaries (ISD) produced by the assembler, FORTRAN compiler, and the linkage editor.

Assembler ISD

The assembler ISD is divided into four sections: a heading, section name table, using tables, and the symbol table (see Figure 23).

HEADING:

Word 1 - Bits 0-15 contain the indicator 4, identifying the ISD as assembler produced.

Word 2 - The length of the ISD in bytes.

Word 3 - Contains a link to the start of the symbol table.

Word 4 - The number of entries in the section name table.

Word 5 - The number of using tables.

Word 6 - The number of entries in the symbol table.

Section Name Tables: The alphameric name and the version identification of each control section (including DSECTs) is entered here, in sequence, by the section number assigned. The name of blank common is represented by eight blank characters; the unlabeled control section is represented by binary zero.

Using Tables: The assembler places a using table in the ISD at every section break and for each USING and DROP statement. All 16 entries are included each time, plus the location at which the table became effective. Registers containing bases for DSECT references are included, but registers containing other external bases are marked as unavailable for checkout purposes.

Symbol Table: The assembler inserts as symbol entries all absolute or simple relocatable value items from its internal dictionary, in addition to entries for each section name. Symbols are grouped according to control section and ordered within each group by ascending location counter value. Immediate value symbols follow those with location counter values.

Name
two words containing the alphameric name of the symbol.

DSECT Flag
contains a 1 bit if this symbol was defined in a DSECT, or if it names a DSECT.

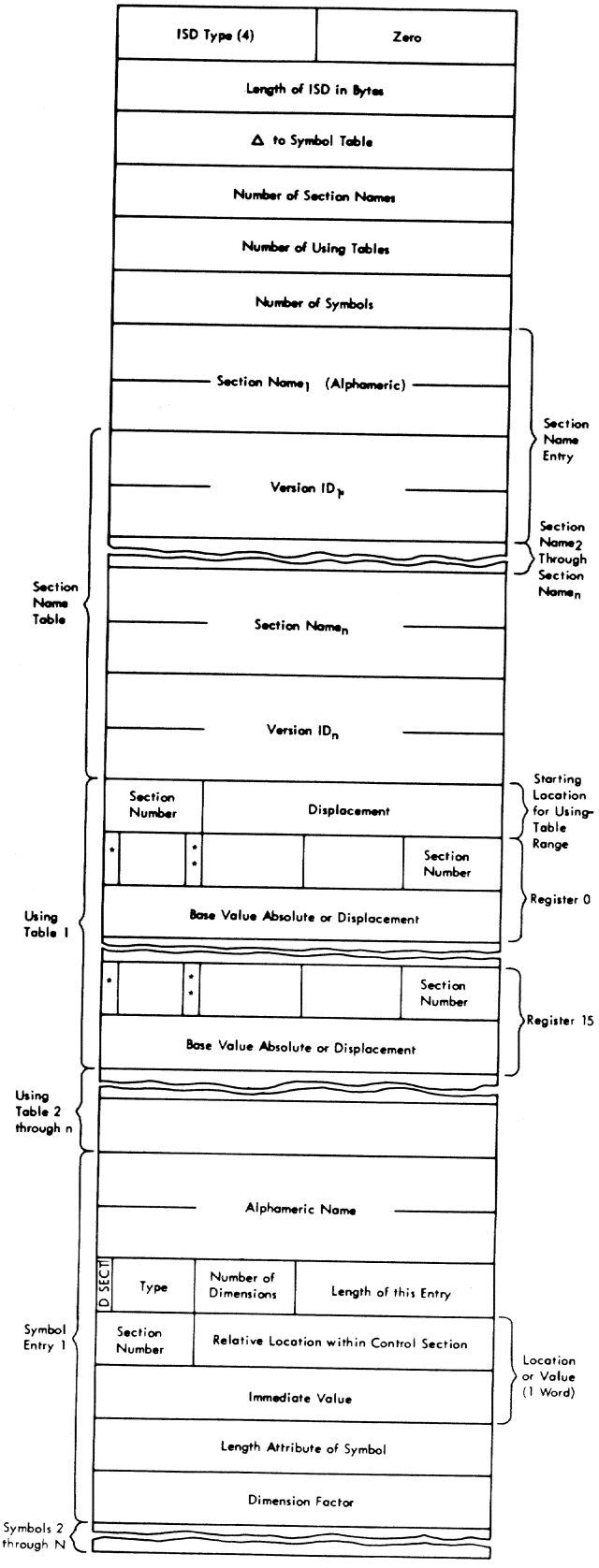


Figure 22. Assembler Internal Symbol Dictionary

Type
identifies the type of field as:

(Assembler Type Attributes)	Code
Instruction (I)	1
Absolute EQUs	2
Section name (J)	3
Integer constant (F,H)	4
Real number (D,E)	5
Character constant (C)	6
Hexadecimal constant (G,K,R,X,M,W,U)	7
Binary constant (B)	8
Packed decimal constant (P)	9
Zoned decimal constant (Z)	10
S-type address constant (S)	11
Other address constant (A,Q,V,Y)	12

Symbols with type attributes T, N, and O are not included in the ISD, nor do undefined symbols appear.

Number of dimensions
has a value of 1 if a duplication factor (other than 1) or multiple contents were used. Otherwise, it has a value of 0.

Length of entry
length in bytes for this symbol entry.

Section number
a number identifying the section in which the symbol was defined. This corresponds to the ordering of the names in the section name table.

Displacement
the location counter value.

Immediate value
if the type was indicated as an absolute EQU, the fourth word of the symbol entry will contain, instead of a section number and displacement, the immediate value of the symbol.

Length
length in bytes of the field defined by this entry.

Dimension factor
this word is included in the symbol entry only if the number of dimensions is non-zero. It contains the byte length of the entire field defined by this entry (i.e., the length times the duplication factor).

FORTRAN ISD

The ISD has four sections: A heading, section name table, statement number table, and a symbol table (see Figure 24).

Heading:
Word 1 - Bits 0-15 contain the indicator 8, identifying the ISD as FORTRAN produced.

Word 2 - The length of the ISD in bytes.

Word 3 - Contains a link to the start of the symbol table.

Word 4 - The number of entries in the section name table.

Word 5 - The number of entries in the statement number table.

Word 6 - The number of entries in the symbol table.

Section Name Table: All control section names and their version identifications (CSECT, PSECT, labeled and blank commons) are listed here. The last two entries are the CSECT and the PSECT.

Statement Number Table: For each executable statement in the program, FORTRAN inserts an entry containing the statement number and the offset from the CSECT base. Entries for unnumbered statements contain a statement number of zero. The entries are arranged in source order.

Symbol Table: The FORTRAN compiler inserts into the symbol table a defining item for all variables, section names, and FORMAT statement numbers. Entries are grouped according to control section and are ordered within each group by ascending location counter value.

Name
two words containing the alphameric name of the variable

Type
bit 1 - set to 1 if the variable is a dummy argument

Type
bits 2-7 - identifies the type of variable as:

Type	Code
Section name	3
Integer	4

Type	Code
Real number	5
Character constant (FORMAT)	6
Complex number	13
Logical	14

Number of dimensions
the number of dimensions of a dimensioned variable (0 for non-dimensioned variables).

Length of entry
length in bytes for this symbol entry.

Section number
a number corresponding to the order of the names in the section table of the ISD.

Displacement
the offset in bytes from the control section base.

Length
length attribute of the variable.

Dimension length
This byte contains the length of the adjustable dimension (2 or 4) which is itself a dummy argument.

Dimension factors
For each dimension of a non-dummy variable array, the dimension product value is listed. The value of the nth dimension factor is the byte length times the product of the sizes of dimensions 1 through n.

For dummy variable arrays, if the dimension is constant, the constant value is stored here. If the dimension is adjustable, the location (displacement from the PSECT base) is stored here.

Linkage Editor ISD

The linkage editor produces, on option, an ISD that permits the PCS to trace back to the original modules (see Figure 25).

The ISD heading contains the output module name, length of the ISD, and total number of input modules, plus two words containing:

1. Link edit level (16 bits) - A counter equal to 1 plus the highest link edit level value present in any of the previously generated

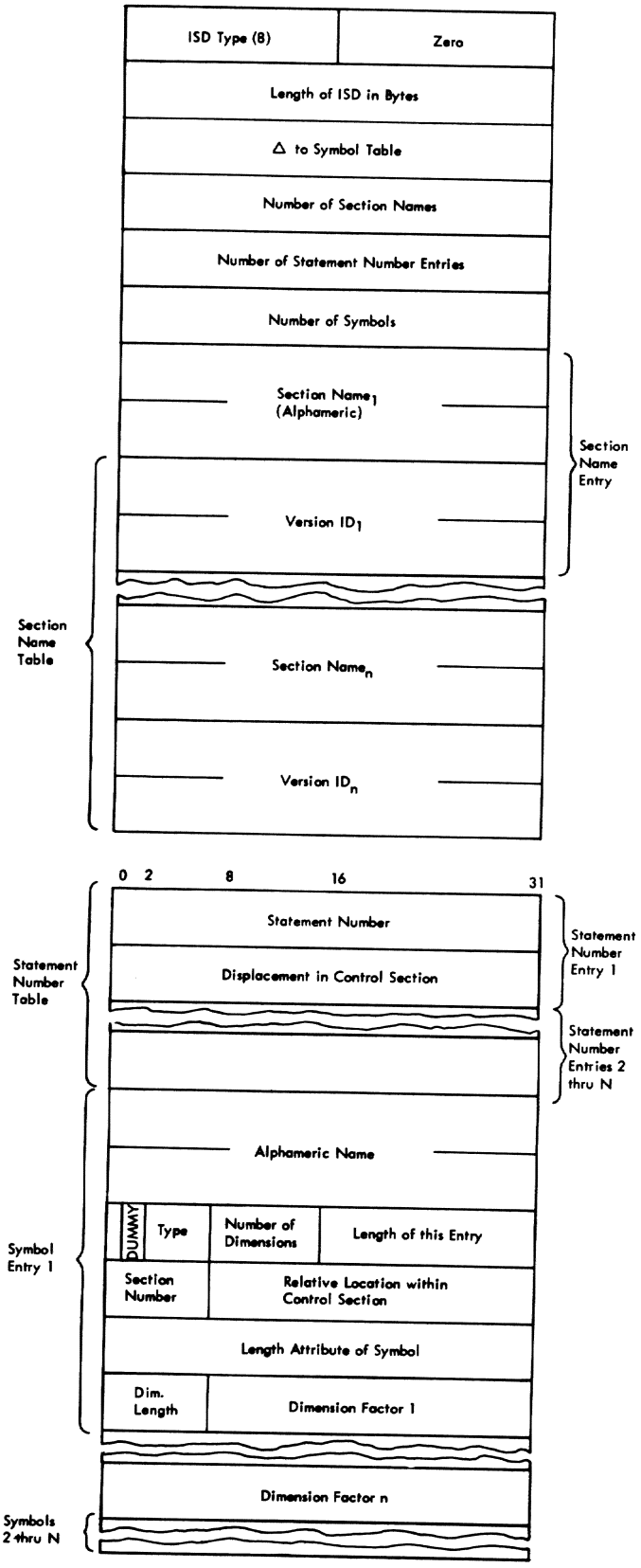


Figure 23. FORTRAN Internal Symbol Dictionary

linkage editor ISDs. If there are no previous linkage editor ISDs present, the value equals 1.

2. Type (16 bits) - 0 for linkage editor ISDs.

3. Displacement to preceding ISD (32 bits) - A displacement (in bytes) from a previously generated linkage editor ISD to the newly generated ISD.

The main body of the dictionary contains a list of entries for each input module. Each input module entry is prefaced by a heading that contains the name of the input module, displacements to the next input module name and to the ISD associated with the module, and the number of output control sections formed. This is followed by a list of entries for each control section from this module; the list of entries is preserved in the output module.

Each control section entry is prefaced by a heading that specifies the output name of the control section and the number of input control sections used to form the output control section. This is followed by a list of the input control sections and their text displacements that were combined or renamed to form this output control section. The simplest case is a list with no entries, which indicates that the output CS is identical to the input CS. If the list contains one entry, it indicates a RENAMED CS. A list with more than one entry indicated a COMBINED CS.

The linkage editor is also responsible for chaining to the ISD it produces all previously generated ISDs associated with the modules input to the linkage editor, thus forming a composite ISD. The ISDs associated with input modules are of two types: those created by a compiler or assembler and those generated by previous output from the linkage editor. The output module from a compilation or assembly contains only one ISD. When two or more such modules are linked by the linkage editor, the associated ISD from each module is retained and added to the composite ISD. The resulting output module, if subsequently input to the linkage editor, causes another ISD to be produced, which is joined to the string of previous ISDs to form a new composite ISD. Thus, the ISD of a module output by the linkage editor contains all the original compiler or assembler produced ISDs, each associated with a former module, plus one or more linkage editor ISDs, one for each module generated by a pass through the linkage editor.

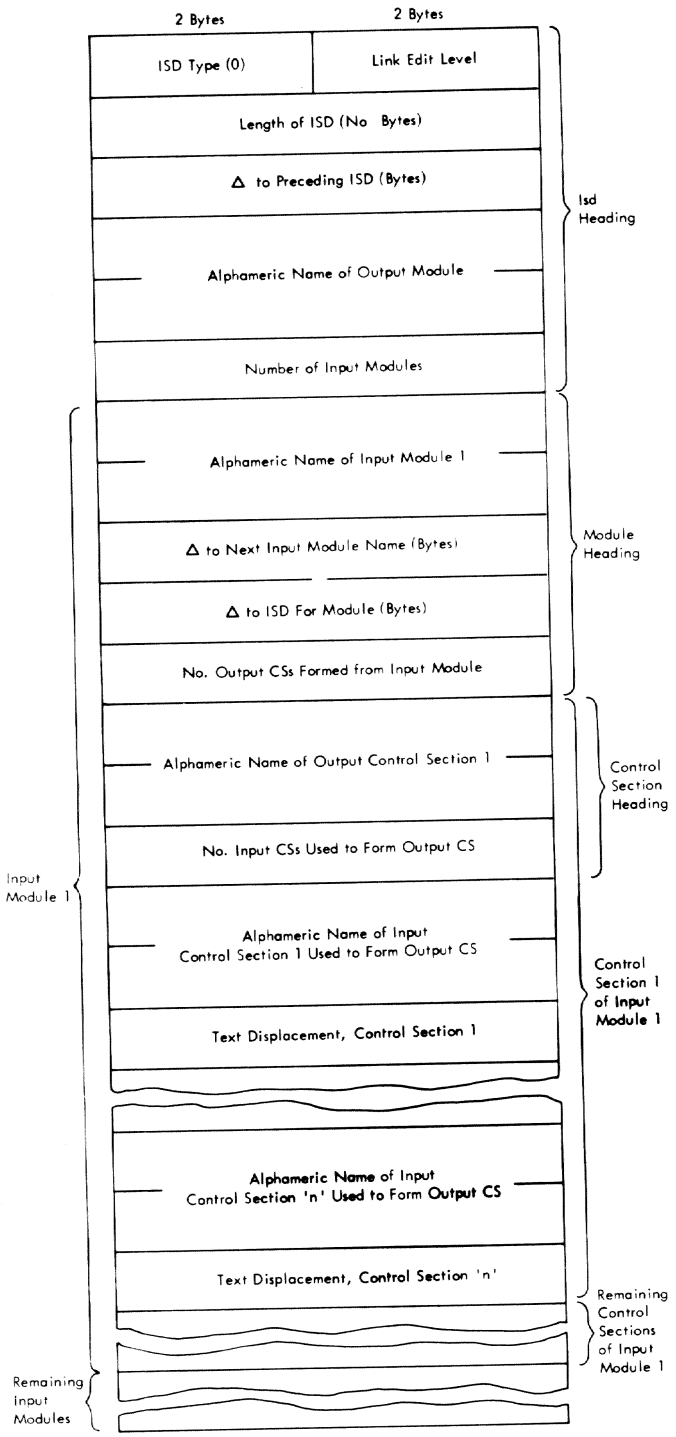


Figure 24. Linkage Editor ISD

The order in which ISDs appear in a program module output by the linkage editor is:

1. The newly generated ISD.
2. All the previously generated ISDs, in the same order in which they were input to the linkage editor.

NEW TASK COMMON (NTC)

New Task Common (NTC) is an area maintained by the system to contain those system values referenced in a single task by more than one command language object module.

PCS uses the following item in New Task Common:

Label	Length	Description
NTCNAM	2 words	This field contains the eight byte name of the current program (the name of the last module assembled, compiled, or loaded).

INTERRUPT STORAGE AREA (ISA)

The Interrupt Storage Area (ISA) contains a complete set of old and new virtual program status words (VPSWs) for all presently defined task interrupts. In addition, the ISA contains space to save general purpose and floating point registers, channel status word and sense data, and other flags and constants.

Since this area is contained in segment 0, page 0 of virtual storage, it may be operated upon by instructions having no base register assignment.

PCS references the user's register and VPSW in the long save area 1 portion of the ISA. These items are:

Label	Length	Description
ISA1I3	1 word	Register 13
ISA1I4	1 word	Register 14
ISA1I5	1 word	Register 15
ISA1I0	1 word	Register 0
ISA1I1	1 word	Register 1
ISA1I2	3 words	Registers 2 - 4
ISA1I5	8 words	Registers 5 - 12
ISA1OP	2 words	User's VPSW
ISA1F0	2 words	FP Register 0
ISA1F2	2 words	FP Register 2
ISA1F4	2 words	FP Register 4
ISA1F6	2 words	FP Register 6

Other ISA items referenced by PCS are:

ISAFA	1 byte	Attention flag A
ISAFB	1 byte	Attention flag B
ISAUTH	1 byte	User authority code
ISARTN	2 bytes	Address of system RTRN routine.
ISATDY	1 word	Pointer to dynamic loader's task dictionary.

SOURCE LIST

PCS uses the following three source list DSECTs:

CHASLP	Source List Page Header
CHASLH	Sublist Header
CHASLM	Source List Marker

These DSECTs are described in the IBM System/360 Time Sharing System: System Control Blocks PLM, Form Y28-2011. They are used to insert parameters and parameter list data for CALL phrases into the source list, and to extract characters from the source list.

COMBINED DICTIONARY ENTRY

PCS uses the combined dictionary (CHADEN) to insert an entry for a command variable. The combined dictionary is described in the IBM System/360 Time Sharing System: System Control Blocks PLM, Form Y28-2011.

PCS COMMUNICATION AREAS AND TABLES

PCS communication areas and tables consist of:

- Location Table (LOCTAB)
- Statement Table (STATAB)
- Internal Symbol Dictionary Map (ISDMAP)
- Source List Item (SLITEM)
- Identified Source List Items (PQNITEM, SQNITEM, SYMITEM, SUBITEM)
- Data Location Item (LOCITEM)
- Data Field Item (FLDITEM)

- Phrase List (PLHEAD)
- Polish String (POLISH)
- Display List (DISPLIST)

ORGANIZATION OF PCS COMMUNICATION AREAS AND TABLES

Tables of fixed maximum length, which contain constants, and which are required throughout PCS processing, are assembled into the appropriate control section; tables of variable length, which are required throughout PCS processing, are allocated storage dynamically.

In some cases, in order to conserve storage, two tables are assigned to the same pages. The Statement Table and the ISD map share the same page. The recomposed instructions share the same page as the Polish string. (For a more detailed discription of page sharing, see the discussion below.)

The various pages dynamically allocated are:

Since any of these tables may require more than one page of storage, a set of pointers is maintained for each.

- A pointer to the first allocated page.
- A pointer to the last permanently allocated page.
- A pointer to the current page.

During processing, the last two pointers may or may not be equivalent. As a statement is being analyzed by PCS input, entries are "temporarily" made to tables. These entries do not become "permanent" until the entire statement has been analyzed and accepted by PCS input. Except for the ISD map, immediate statements never cause permanent entries to tables.

If a table overflows to an additional page while temporary entries are being made, that page is not considered permanent until the entire dynamic statement is accepted.

SPACE SHARING BETWEEN TABLES

As discussed earlier, the ISD map and STATAB share space in the same pages of storage. STATAB entries are made from the bottom of the page (position 0) consecutively to the top (position 4095). ISD map entries are made from the top of the page, consecutively toward the bottom.

A sample STATAB/ISDMAP page might appear as in Figure 26, below.

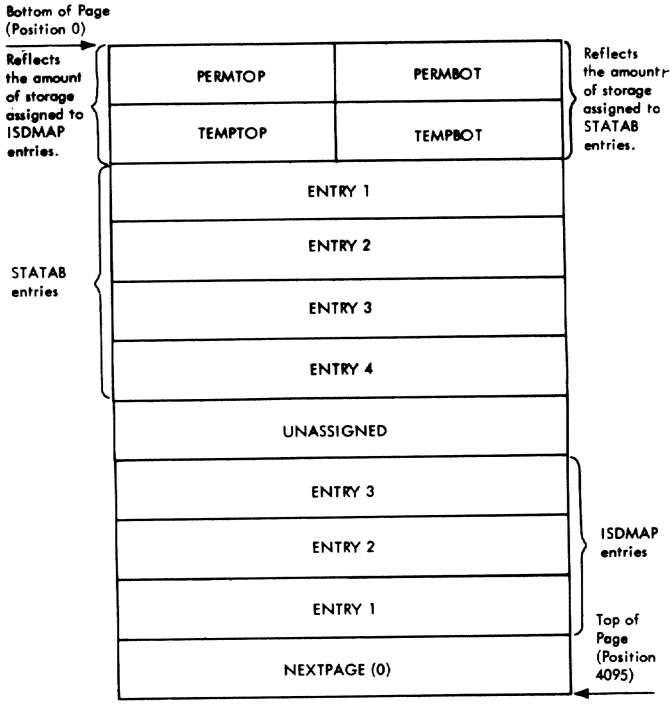
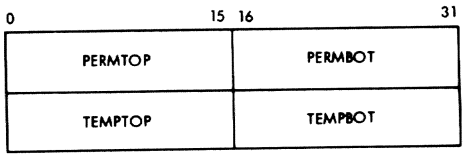


Figure 25. Sample STATAB/ISDMAP Page

To maintain control over the amount of space allocated in each page, three words of the page are reserved for page control information. The first two words of the page are the page header and have the form:



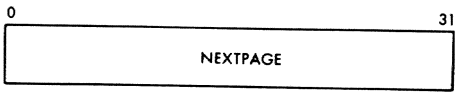
PERMTOP
index into the page. Contains the number of bytes of permanently assigned storage preceding the page trailer.

PERMBOT
index into the page. Contains the number of bytes of permanently assigned storage following the page header.

TEMPTOP
index into the page. Contains the number of bytes of storage assigned preceding the page trailer.

TEMPBOT
index into the page. Contains the number of bytes of storage assigned following the page header.

The last word of the page is the page trailer, and has the form:



NEXTPAGE
contains the virtual storage address of the next page for this table(s).

When PCS searches STATAB for a particular entry, the statement number of the entry is multiplied by 8 (the STATAB entry length, in bytes) to compute a byte index. The byte index is then compared to the STATAB length on the first page (as contained in PERMBOT). If the byte index is greater than PERMBOT, it is decremented to obtain a new byte index which is compared to the PERMBOT of the next STATAB page. This process continues until the byte index is less than or equal to the PERMBOT of a page. At this point, the byte index is added to the virtual storage address of that STATAB page to obtain the address of the particular STATAB entry being sought.

When PCS searches the ISD map for a particular entry by name, the search proceeds through the page from the top toward the bottom, because the entries were stored in inverted order. Each entry in the ISD map is compared to the search name. The search is terminated when either the ISD map entry is found, or the end of the ISD map is detected. The end of the ISD map is indicated when the overflow pointer to the next page is zero.

To locate an ISD map entry by number, the entry number is multiplied by the entry length (16 bytes) to compute the byte index.

The byte index is then compared to the ISD map length of the first page (as contained in PERMTOP). If the byte offset is greater than PERMTOP, it is decremented by the value of PERMTOP, and the result is compared to the PERMTOP of the next page. This process is repeated until the byte offset is less than, or equal to, the PERMTOP of a page. At this point, the byte index is subtracted from the top of the page minus four bytes plus one byte, to obtain the address of the particular ISD map entry being sought. For example, to find the displacement address in the page, of ISD map entry one, the following figures would apply.

ISD map entry number =	1	4095	Top of Page displacement
Times entry length =	16	-4	Length of NEXTPAGE indicator
	16 →	-16	Byte index
		+1	Compensation for zero origin
		4076	Displacement from bottom of page (Displacement zero)

In the example above, notice that any entry may be found by varying the ISD map entry number which computes the byte index.

The recomposed instructions and Polish string also share the same pages, although in this case entries are made for both types, from the bottom of the page, toward the top. PERMTOP and TEMPTOP will always contain zero for these pages.

PCS input makes entries for the Polish string. These entries are always temporary, and their storage assignments are reflected in TEMPBOT. When PCS input terminates processing, storage assigned to the Polish string is released.

PCS output makes entries for the recomposed instructions. These entries are always considered "permanent" in that the storage assignment for this table is reflected in both PERMBOT and TEMPBOT. Each recomposed instruction is followed by a PCSVC. When PCS output regains control as a result of having executed this PCSVC, the storage assigned to the recomposed instruction is released.

The pages for the phrase list are not shared. Entries are made consecutively from the bottom of the page toward the top. PERMTOP and TEMPTOP will always be zero for these pages.

Page control information is maintained for all generated code pages. Code is generated from the bottom of the page toward the top, and its storage allocation is reflected by PERMBOT and TEMPBOT. Inversely, constant and base register data is assigned storage from the top of the page toward the bottom, with the storage assignments being reflected by PERMTOP and TEMPTOP. The page trailer word for linking overflow pages is maintained. It may be loaded into register 15 during the execution of generated code to provide code cover for the execution of subsequent generated code.

The location table (LOCTAB) is an exception to the page control method described. Entries are not made in the table in any sequential pattern, but may be scattered throughout the page. A hashing algorithm is used to search the location table pages for a particular entry. The hash result furnishes a chain of eight entries per page, each of which is inspected during a search. The search is terminated when either the applicable entry is found, or a null entry is found. Since each LOCTAB entry is in many hash chains, all entries made in the location table are considered permanent. When a LOCTAB entry is no longer required, its storage allocation is made available for reuse.

If the search through the location table for a particular entry is unsuccessful, and 'entry available for reuse' entry will be used in preference to a null entry.

LOCATION TABLE (LOCTAB)

The location table (LOCTAB) (Figure 27), contains an entry for each PCSVC stored in the user's program as a result of processing a dynamic statement. LOCTAB also contains an entry for each PCSVC that follows a recomposed instruction.

A hashing algorithm, based on the address of the PCSVC, is used to determine the relative position of an entry in the table. The same algorithm is used to efficiently search the table for a particular entry.

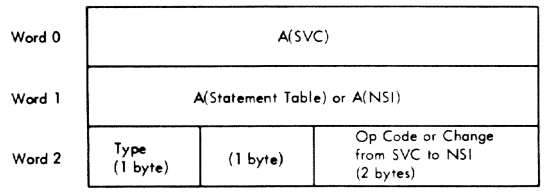


Figure 26. Location Table (LOCTAB) Entry

A(SVC) is the virtual storage address of the byte following the PCSVC that is associated with the LOCTAB entry.

A(Statement Table) is a full word address pointing to the first Statement Table entry associated with this location, for a LOCTAB entry identified as IN USE.

A(NSI) is the address of the next sequential instruction, following the SVC in the user's program, for a LOCTAB entry identified as RETURN.

TYPE is a one byte indicator which designates the type of entry. The possible codes and their meanings are:

00 - NULL	This entry has never been used.
01 - LTINUSE	This entry is for a PCSVC which is stored in the user's program.
04 - LTRTN	This entry is for a PCSVC that follows a recomposed instruction.
2C - REMOVE	This entry is available for reuse.

Op Code specifies the two bytes in the user's program which were replaced by the PCSVC for a LOCTAB entry identified as IN USE.

Change from SVC to NSI. For a return code entry, this halfword contains the byte length of the recomposed instruction.

Initially, all LOCTAB entries are identified as NULL.

PCS input forms a LOCTAB entry identified as IN USE for the first occurrence of a specified AT operand. This LOCTAB entry is associated with the

specified AT operand in subsequent AT phrases via the AT phrase list.

PCS output forms a LOCTAB entry identified as RETURN, for each PCSVC that follows a recomposed instruction.

When PCS Output gains control as a result of having executed a PCSVC which follows a recomposed instruction, the LOCTAB entry is cleared and identified as REMOVE. If a user removes all dynamic statements for a particular location, the instruction which was overlaid by the PCSVC is restored in the user's program, thereby nullifying the PCSVC, and the LOCTAB entry for the PCSVC is cleared and identified as REMOVE.

STATEMENT TABLE (STATAB)

The Statement Table (STATAB) (Figure 28), contains an entry for each dynamic statement entered by the user. The Statement Table entry for immediate statements is contained in the PCS PSECT.

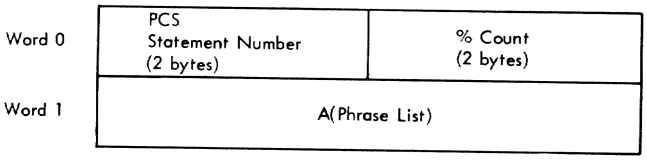


Figure 27. Statement Table (STATAB) Entry

PCS Statement Number is a halfword integer assigned to dynamic statements by PCS for identification purposes. The first statement number assigned, is one.

% Count is a halfword integer which is incremented by PCS Output each time the statement entry is processed. It is initialized to zero, and incremented before processing the phrase list associated with the entry.

A(Phrase List) is the full word address of the first phrase list formed for the statement.

PCS Input Phase I forms a phrase list for the phrase being processed. This phrase list is linked to the immediate STATAB entry in the PSECT. If the current phrase is an AT phrase, the statement is considered dynamic and the immediate STATAB entry is assigned temporary storage in the last statement table page.

PCS Input Phase II locates the STATAB entry being processed and initializes the % count to zero. If the statement is dynamic, the storage for the STATAB entry is permanently assigned. The Statement Number in the immediate STATAB entry becomes output for identification purposes, and is then incremented to prepare for the next dynamic statement.

INTERNAL SYMBOL DICTIONARY MAP (ISDMAP)

The ISD map (Figure 29), contains an entry for each ISD loaded. It is generated by PCS input and used by the PCS output and DISPLAY/DUMP components.

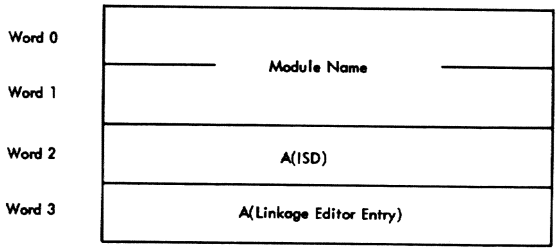


Figure 28. Internal Symbol Dictionary (ISDMAP) Entry

Module Name is an assembled, compiled, or link edited module name.

A(ISD) is the virtual storage address of the Internal Symbol Dictionary.

A(Linkage Editor Entry) is the virtual storage address of the primary qualifier's ISDMAP entry. In an ISDMAP entry for a primary qualifier, this word contains a zero.

When the ISD for a module is loaded, an ISDMAP entry for a primary qualifier is made. If the ISD is a link edited ISD, an ISDMAP entry is made for each assembled or compiled module.

For example, assume that two ISDs have been loaded, one for a module called PGMONE and the other for a module called PGMTWO. Further assume that PGMTWO is a link edited module that was created by combining three other modules named PGMA, PGMB, and PGMC. The ISDMAP organization, with the entries made in inverted order, would appear as shown in Figure 30.

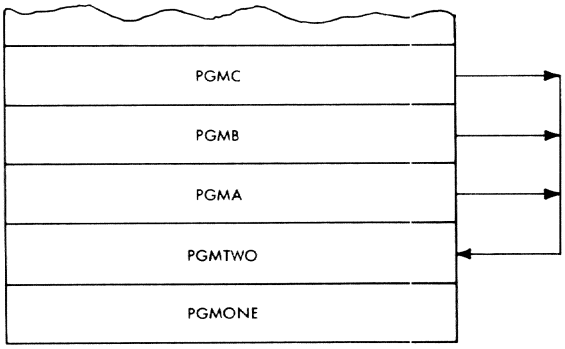


Figure 29. Sample ISDMAP

SOURCE LIST ITEM (SLITEM)

A source list item is formed for each character string scanned (SCANFLD). A source list item has the format of Figure 31.

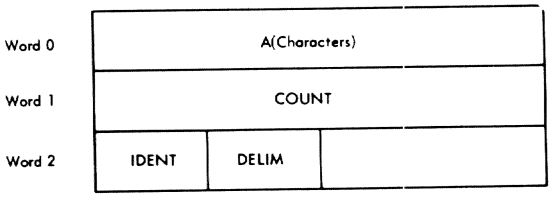


Figure 30. Source List Item (SLITEM)

A(Characters) is the address of the next available byte in a buffer in the PCS PSECT.

COUNT is the number of characters, not including delimiters, extracted from the source list and stored in the buffer.

IDENT is a one byte indicator identifying the character string scanned. The possible codes and their meanings are:

00 - NULL	The first character scanned is the delimiter.
04 - NUMBER	The entire string consists of the characters 0 through 9.
08 and 0C - ALPHA	This string contains one or more non-numeric characters.

Leading and trailing blanks are ignored. An embedded blank is treated as a non-numeric character.

DELIM is the character string delimiter. The various delimiters are:

+ plus	- minus
* multiply	/ divide
& and	| or
> greater than	< less than
= equal to	¬ not
(left parenthesis	) right parenthesis
' quote	. period
: colon	; semicolon
, comma	end-of-block

The letters D, E, and R are treated as delimiters in numeric fields. The delimiter does not participate in the identification of the character string.

The source list contains a string of characters. This string is not necessarily contiguous because of synonym substitution, line continuation and/or procedure expansion. Characters are extracted from the source list (GETCHAR) and stored in the buffer in the PSECT to provide a continuous character string.

When a source list item has been formed, the synonym checks are performed. These checks are:

- The string must be identified as ALPHA.
- The string must be 8 characters or less in length.
- A synonym is possible at this point (i.e., the SYNONYM indicator is set).

If all the synonym tests are successful, and the character string has a synonym, the delimiter is restored to the source list and the character string and delimiter are eliminated from the continuous character string buffer. The source list is expanded to include the synonym character string, and a source list item for the first character string of the synonym is formed.

Because of the syntax of PCS, a source list item may be formed prematurely. When this happens, the character string and delimiter of the source list item are "eliminated" from the continuous character string buffer. An

indicator (FORMED), is set to indicate that a source list item is still outstanding. When the next source list item is requested, the character string and delimiter of the outstanding source list item are restrored to the continuous character string buffer, the FORMED indicator is cleared, and the outstanding source list ierm is returned.

IDENTIFIED SOURCE LIST ITEMS (PQNITEM, SQNITEM, SYMITEM, SUBITEM)

In the process of identifying a data location, several source list items may be formed. When a source list item is formed, the IDENT and delimiter are inspected and the source list item is moved to one of the four identified source list items. These identified source list items have the same format as the source list item (SLITEM). The processing of identified source list items varies according to the data location identified.

Explicitly Qualified Internal Symbols

The identified source list items, PQNITEM and SQNITEM (Figure 32), specify the primary and secondary qualifying module names. VALMOD processes these items and modifies the item format.

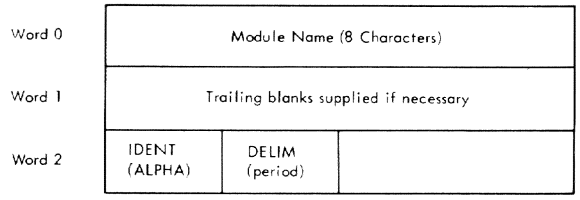


Figure 31. Qualifying Name Items

Internal and External Symbols

The identified source list item, SYMITEM (Figure 33), specifies the symbol name. DATALOC modifies the item format.

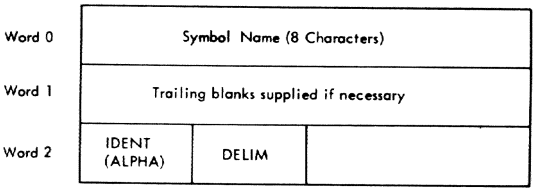


Figure 32. Symbol Name Item

DATALOC checks the symbol name for one of three special strings; %, %COM, or %CSECT.

If the symbol name is %, a data location item for the dynamic count is formed.

If the symbol name is %COM, it is considered to be an alias for Blank Common and is changed to all blanks.

If the symbol name is %CSECT, it is considered to be an alias for an unnamed control section and is converted to all zeros. Since unnamed control sections must be internal symbols, the symbol is identified as being implicitly qualified.

The symbol name item is presented to VALSYM for evaluation as an internal symbol. If the internal symbol evaluation is not successful, the symbol name is presented to EXTERNAL for evaluation as an external symbol.

Statement Numbers

The identified source list items, SYMITEM (Figure 34) and SUBITEM (Figure 35), specify the statement number and subscript. DATALOC converts the statement number and subscript to integer and modifies the item formats.

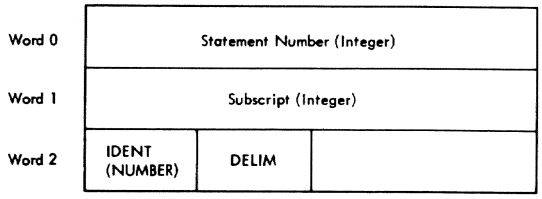


Figure 33. Symbol Name Item for Statement Number

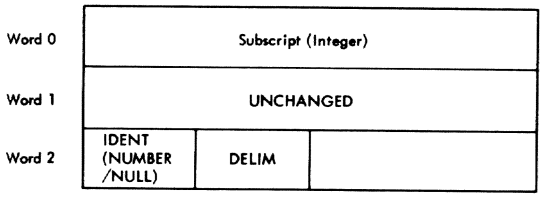


Figure 34. Subscript Item

The symbol name item is presented to VALSYM for evaluation as a statement number. The statement number is used to search to statement number table and the subscript is used as an index which is applied to the entry found. If an

entry in the statement number table is not found, VALSYM evaluates the symbol name item as an internal symbol. This evaluation must be successful or a diagnostic will be formed.

Floating Point Constants

All four identified source list items are used to express a floating point constant. Their contents represent the following portions of the number:

SYMITEM represents the integer portion of the floating point constant.

SUBITEM represents the fractional portion.

PQNITEM represents the sign of the exponent.

SQNITEM represents the exponent.

SYMITEM and SUBITEM are converted to integer, a character at a time. (The string length of SYMITEM will have been previously adjusted to eliminate leading zeros.) The result is stored in CONAREA as an unnormalized, double precision, floating point number (1 byte exponent and 7 byte integer). The exponent portion of the number is then converted to integer and adjusted to include the sign. Exponent range checks are made, and the exponent is adjusted to account for the fraction. The resulting floating point number is then multiplied or divided by the appropriate power of 10, and the result is stored in CONAREA.

Other Data Locations

All other data locations are specified by SYMITEM. Processing of this identified source list item for the other data locations, is described in the DATALOC subroutine description.

DATA LOCATION ITEM (LOCITEM)

A data location item (LOCITEM) (Figure 36), is identified by DATALOC. It has the format illustrated below:

LOCITEM is the address of the next available byte in the continuous character string buffer. It is set by DATALOC in initializing LOCITEM. This pointer is used for diagnostic purposes.

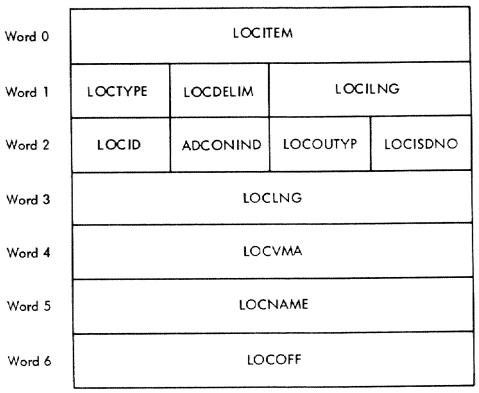


Figure 35. Data Location Item (LOCITEM)

LOCTYPE is a one byte indicator which designates the data type of the data location. The possible codes and their meanings are:

00 - UNKNOWN Type is undefined. This is set by DATALOC in initializing LOCITEM.

04 - ISDINT Type is integer. This is set by DATALOC for integer constants and % count.

05 - ISDREL Type is real. This is set by DATALOC for floating point constants and floating point registers.

06 - ISDCHC Type is character. This is set by DATALOC for character constants.

07 - ISDHEX Type is hexadecimal. This is set by DATALOC for hexadecimal constants.

ISDSTP is the data type of an internal symbol, as specified in the ISD. It is set by VALSYM.

DENCOD is the entry code for a command variable in the combined dictionary. It is adjusted and set by EXTERNAL.

LOCDELIM is the delimiter of the data location character string. In most cases, this character is the de-

limiter of SLITEM. Prior to exiting, DATALOC moves the SLITEM delimiter to LOCDELIM. If a source list item is still outstanding, (i.e., FORMED is set), the delimiter is already in LOCDELIM. An end-of-block delimiter is converted to a semicolon. Valid LOCDELIMs are:

+ plus	- minus
* multiply	/ divide
& and	| or
> greater than	< less than
= equal to	¬ not
(left parenthesis	) right parenthesis
: colon	; semicolon
, comma	' quote (only if ADCONIND is set)

LOCILNG indicates the item length. It is set to the number of bytes in this item following LOCID. It is used by DATAFLD in the formation of a data field item for a range.

LOCID is a one byte indicator of the identification of the data location as determined by DATALOC. This identification is subject to modification during the evaluation process. The possible codes and their meanings are:

00 - NULL	Identifies an operator/delimiter. This identificaiton is set by VALSYM to signify an unsuccessful evaluation of an internal symbol. A NULL identification nullified the INTERN identification.
04 - GENERAL	Identifies a general register.
08 - SINGLE	Identifies a single precision register.
0C - DOUBLE	Identifies a double precision register.
10 - PERCNT	Identifies the percent count (%).
14 - EXTERN	Identifies an external symbol.
18 - INTERN	Identifies an internal symbol. VALSYM sets this identification (thereby nullifying the STATNO identification) if the evaluation of a statement number is unsuccessful.
1C - STATNO	Identifies a statement number.
20 - ARRAY	Identifies a subscripted array.
24 - ADDRES	Identifies a hexadecimal address.
28 - HCON	Identifies a hexadecimal constant.
2C - COMVAR	Identifies a defined command variable. This identification is set by EXTERNAL.
30 - UNDEFINE	Identifies an undefined command variable. This identification is set by EXTERNAL.
34 - ERROR	Identifies a syntax error.
38 - ICON	Identifies an integer constant.
3C - CCON	Identifies a character constant.
40 - ACON	Identifies an address constant. This identification is used in checking the context of the address constant. If the context is valid, this identification is modified to that of the symbol specified in the address constant.
44 - FCON	Identifies a floating point constant.
80 - SOFFSET	Identifies a data location with offset. This bit is ORed into LOCID.

ADCONIND is one byte which is set to indicate that an address constant is being processed.

LOCOUTYP is a byte containing a code which is the same as LOCTYPE, indicating the user's choice of output with an offset field. The first half-byte may also contain information regarding special forms of output as follows:

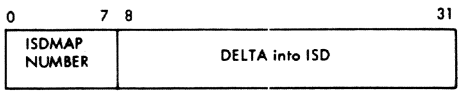
code 10 - user has requested the identity of an address, i.e. the CSECT name that contains that address, by means of the ID? parameter.

LOCISDNO is a byte containing the number of an ISD map member for a module that has been the subject of a request for a display/dump in symbolic format, where the module was identified using an external symbol with an offset.

LOCLNG contains the data length of the variable or constant. For internal symbols, the data length is obtained from the ISD. For command variables, the data length is obtained from the combined dictionary. The length of all other data locations is implicitly defined. If the data location is offset, an explicit length can be specified and will be used in place of the defined length.

LOCVMA is the virtual storage address of the first byte of a variable. For registers, this word contains the ISA address. For undefined command variables, the eight bytes to LOCVMA and LOCNAME contain the command variable name.

LOCNAME. For external symbols, this word contains the address of the DEF entry in the CSD of the PMD. For defined command variables, this word contains the address of the combined dictionary entry. For internal symbols, statement numbers, and subscripted arrays, this word has the form:



and where: byte 1 contains the entry number in the ISDMAP for the ISD used in evaluating the data location.

bytes 2-4 contain the index from the base address of the ISD to the defining ISD entry.

LOCOFF is the address of the Polish string formed for a subscript/offset. It is set by SUBPOL.

DATA FIELD ITEM (FLDITEM)

A data field item (FLDITEM) (Figure 37), is formed by DATAFLD. DATALOC is called to form a data location item which is then moved to the data field item. If a range is specified, DATALOC is again called to form a second data location item, which is then combined with the data field item.

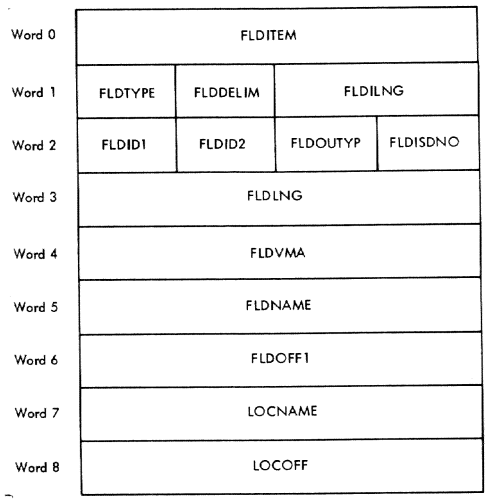


Figure 36. Data Field Item (FLDITEM)

FLDITEM contains the same address as LOCITEM for the first data location item.

FLDTYPE contains LOCTYPE if a single data location item is formed (i.e., not a range). If a range is specified, FLDTYPE is set to 00 (UNKNOWN). If a range of registers is identified, FLDTYPE is set to LOCTYPE of the second data location item.

FLDDELIM is set to LOCDELIM of the last data location item formed.

FLDILNG is the data field item length. It is set to the number of bytes in this item following FLDID1. It is used by the phrase list control routines for the allocation of storage for phrase list entries.

FLDID1 and FLDID2. FLDID1 is the LOCID of the first data location item. If a second data location item is formed, the LOCID of that item is placed in FLDID2. Otherwise FLDID2 is set to 00 (NULL). DATAFLD modifies FLDID1 in the process of forming the data field item. The possible codes for FLDID1, with the corresponding codes for FLDID2, and their meanings are:

00 - NULL Identifies an operator/delimiter. FLDID2 must also be NULL.

04 - GENERAL — Identifies a general register. DATAFLD sets this to the LOCID of the last data location item formed. Register range notations of the form 0:4R and 0R:4R are equivalent. FLDID2 must be either NULL or GENERAL.

08 - SINGLE — Identifies a single precision register. DATAFLD sets this to the LOCID of the last data location item formed. Register range notations of the form 0:4E and 0E:4E are equivalent. FLDID2 must be NULL or SINGLE.

0C - DOUBLE — Identifies a double precision register. DATAFLD sets this to the LOCID of the last data location item formed. Register range notations of the form 0:4D and 0D:4D are equivalent. FLDID2 must be NULL or DOUBLE.

10 - PERCNT — Identifies the percent count (%). FLDID2 must be NULL.

14 - EXTERN — Identifies an external symbol. FLDID2 must be NULL or EXTERN.

18 - INTERN — Identifies an internal symbol. FLDID2 must be either NULL, INTERN or ARRAY.

1C - STATNO — Identifies a statement number. FLDID2 must be NULL or STATNO.

20 - ARRAY — Identifies a subscripted array. FLDID2 must be either NULL, INTERN or ARRAY.

24 - ADDRES — Identifies a hexadecimal address. FLDID2 must be NULL or ADDRES.

28 - CONST — Identified a constant. This identification applies to data location items identified as HCON, ICON, CCON, or FCON. FLDID2 must be NULL.

2C - COMVAR — Identifies a defined command variable. FLDID2 must be NULL.

30 - UNDEFINE — Identifies an undefined command variable. FLDID2 must be NULL.

34 - ERROR — Identifies a syntax/context error.

FLDOUTYP contains LOCOUTYP. In the case of a range the second LOCOUTYP will become FLDOUTYP.

FLDISDNO contains LOCISDNO. In the case of a range the second LOCISDNO will become FLDISDNO.

FLDLNG contains the length of the data field. If a single data location item was formed, FLDLNG contains the LOCLNG of that item. If two data location items were formed, it contains the difference between the LOCVMAs of the items. In data field items for a single register, the register number is stored in the third and fourth bytes of FLDLNG. In data field items for register ranges, the first register number is stored in the third byte of FLDLNG and the second register is stored in the fourth byte.

FLDVMA is the virtual storage address of the first byte of the data field (i.e., LOCVMA of the first data location item).

FLDNAME is the LOCNAME of the first data location item.

FLDOFF1 is the address of the Polish string formed for the subscript/offset of the first data location item.

LOCNAME is the LOCNAME of the second data location item.

LOCOFF is the address of the Polish string formed for the subscript/offset of the second data location item.

PHRASE LIST (PLHEAD)

PLIDENT identifies the current phrase. The possible codes and their meanings are:

08 - SET — Identifies this as a SET phrase.

0C - DISPLAY Identifies this as a DISPLAY phrase.

10 - DUMP Identifies this as a DUMP phrase.

14 - CALL Identifies this as a CALL phrase.

18 - BRANCH Identifies this as a BRANCH phrase.

1C - GO Identifies this as a GO phrase.

20 - IF Identifies this as an IF phrase.

24 - STOP Identifies this as a STOP phrase.

28 - AT Identifies this as an AT phrase.

PLQUAL is the ISDMAP entry number for the ISD used for automatic qualification of implicitly qualified internal symbols.

PLDELTA is the byte length of the phrase list. It is initially set at 4 (the length of the phrase list header).

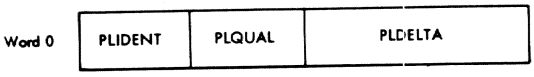


Figure 37. Phrase List Header (PLHEAD)

Phrase List Processing - Phase I

PCS Input (Phase I) forms a phrase list header (PLHEAD) (Figure 38) for the current phrase. Storage for the phrase list header is preallocated, while storage for phrase list entries is allocated by the phrase list control routines. As the entries are inserted in the list, PLDELTA is incremented to reflect the current length of the phrase list, in bytes. When the current list has been completely formed, PCS Input (Phase II) allocates storage for a phrase list terminator (Figure 39). If the statement associated with this phrase list is dynamic, the next phrase list header will be formed and will overlay the current terminator. This procedure results in the automatic linking of phrase lists in dynamic statements. This logic has no effect on phrase lists associated with immediate statements, since all assigned storage is released following execution of the immediate phrase.

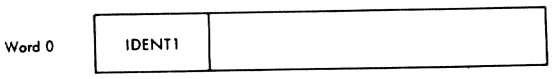


Figure 38. Phrase List Terminator/ Continuation Trailer

IDENT1 contains the identification of this word. Possible codes and their meanings are:

00 - TERMINAT Identifies this as a phrase list terminator.

04 - CONTINUE Identifies this as a continuation trailer.

During the formation of an AT, DISPLAY, DUMP, or SET phrase list, a page overflow may occur during the phrase list storage allocation process. In this case, a continuation trailer (Figure 38), is formed at the end of the list. The phrase list is expanded by a page and PLDELTA is set to 4 thus reinitializing the phrase list header. The entry which caused the overflow is then inserted in the phrase list.

To minimize page referencing and/or phrase list storage allocation, the following exceptions should be noted:

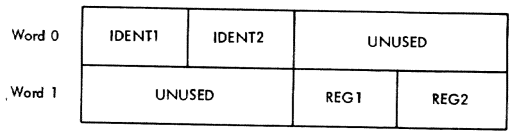
1. A BRANCH, GO, or STOP phrase list is an implicit phrase list terminator. A termination trailer is not formed for these lists.

2. A BRANCH, CALL, IF, GO, or STOP phrase list for an immediate phrase is formed in the PCS PSECT and is not inserted in the phrase list pages.

Phrase list entry insertion is governed by the phrase list control routines. The formats of these entries are unique for each control routine. These control routines are discussed below.

DISPLAY/DUMP CONTROL ROUTINE: An entry is made in the phrase list for each operand in the DISPLAY/DUMP phrase. The format of each entry is determined by the syntax used to express the operand and the process used to evaluate it.

Register:



IDENT1 identifies the type of register entry. The possible codes and their meanings are:

04 - GENERAL Identifies a general register.

08 - SINGLE Identifies a single precision register.

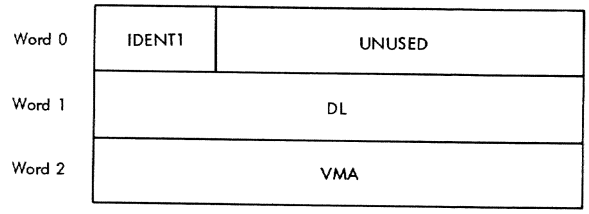
0C - DOUBLE Identifies a double precision register.

IDENT2 identifies the type of register entry when a register range has been specified. It contains the same codes as IDENT1. If the entry is for a single register (i.e., no range), IDENT2 will contain code 00 (NULL).

REG1 contains the number of the first register.

REG2 contains the number of the last register specified in a range. If no range is specified, this entry is the same as REG1.

% Count:



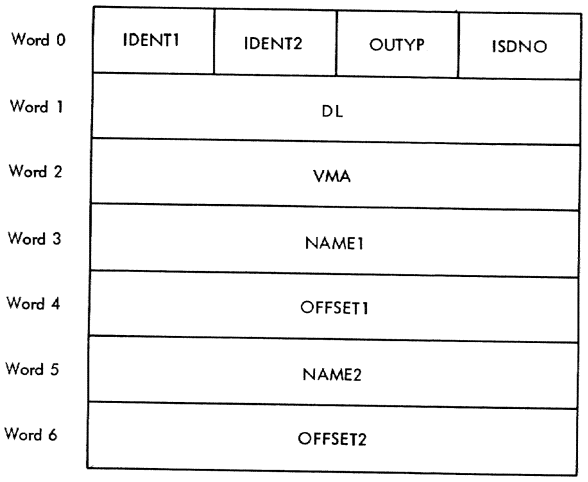
IDENT1 identifies the type of entry. The code for a % count entry is:

10 - PERCNT Identifies the dynamic count of the statement.

DL is a word field containing the data length.

VMA is the virtual storage address of the dynamic count in the current STATAB entry.

External Symbol:



IDENT1 identifies the type of entry. The possible codes for an external symbol, and their meanings are:

14 - EXTERN Identifies an external symbol.

94 - EXTERN+ SOFFSET Identifies an external symbol with offset.

IDENT2 identifies the existence of an external symbol range. The possible codes and their meanings are:

00 - NULL No range specified.

14 - EXTERN Identifies an external symbol range.

94 - EXTERN+ SOFFSET Identifies an external symbol range with offset.

OUTYP contains FLDOUTYP.

ISDNO contains FLDISDNO.

DL contains the number of bytes to be displayed/dumped.

VMA contains the virtual storage address of the first external symbol.

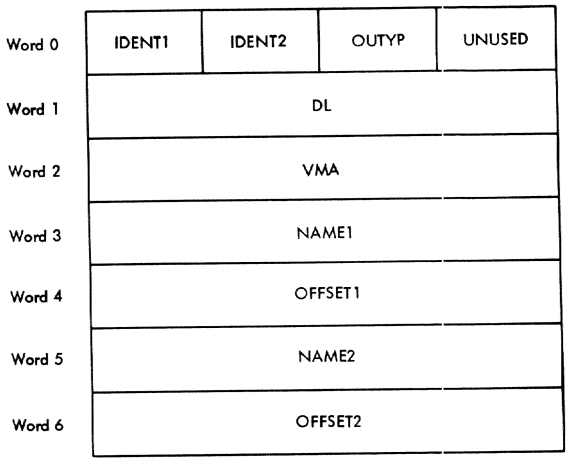
NAME1 contains the address of the first symbol definition in the PMD.

OFFSET1 contains the address of the Polish string formed for the offset. This word is present only if IDENT1 contains code 94 (EXTERN+SOFFSET).

NAME2 contains the address of the second symbol definition in the PMD. This word is present if IDENT2 is not NULL.

OFFSET2 contains the address of the Polish string formed for the offset of the second symbol. This word is present only if IDENT2 contains code 94 (EXTERN+SOFFSET).

Internal Symbols and Subscripted Arrays:



IDENT1 identifies the type of entry. The possible codes for an internal symbol or subscripted array entry, and their meanings are:

18	INTERN	Identifies an internal symbol.
20	ARRAY	Identifies a subscripted array.
98	INTERN+ SOFFSET	Identifies an internal symbol with offset.

IDENT2 identifies the existence of an internal symbol range. The possible codes and their meanings are:

00	NULL	No range specified.
18	INTERN	Identifies an internal symbol range.
20	ARRAY	Identifies an internal symbol range.
98	INTERN+ SOFFSET	Identifies an internal symbol range.

OUTYP contains FLDOUTYP.

DL contains the number of bytes to be displayed/dumped.

VMA contains the virtual storage address of the first symbol entry.

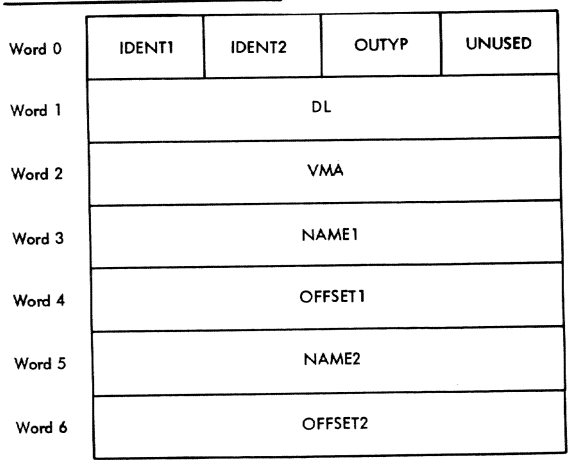
NAME1 contains the ISDMAP entry number for an ISD and an index into the ISD to locate the first symbol entry.

OFFSET1 contains the address of the Polish string formed for the offset/subscript. This word is present if IDENT1 contains either code 20 (ARRAY) or 98 (INTERN+SOFFSET).

NAME2 contains the ISDMAP entry number for an ISD and an index into the ISD to locate the second symbol entry. This word is present if IDENT2 is not NULL.

OFFSET2 contains the address of the Polish string formed for the subscript/offset of the second symbol. This word is present if IDENT2 contains either code 20 (ARRAY) or 98 (INTERN+SOFFSET).

Statement Number:



IDENT1 identifies the type of entry. The possible codes for a statement number entry, and their meanings are:

1C	STATNO	Identifies a statement number.
9C	STATNO+ SOFFSET	Identifies a statement number with offset.

IDENT2 identifies the existence of a statement number range. The possible codes and their meanings are:

00	NULL	No range specified.
1C	STATNO	Identifies a statement number range.
9C	STATNO+ SOFFSET	Identifies a statement number range with offset.

OUTYP contains FLDOUTYP.

DL contains the number of bytes to be displayed/dumped.

VMA contains the virtual storage address of the first statement number.

NAME1 contains the ISDMAP entry number for an ISD and an index into the ISD

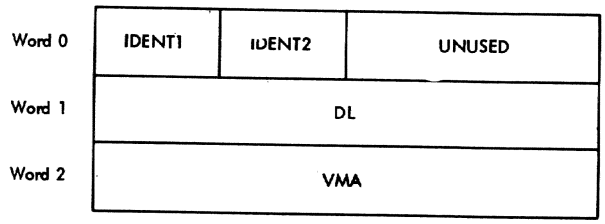
to locate the first statement number entry.

OFFSET1 contains the address of the Polish string formed for the offset. This word is present if IDENT1 contains code 9C (STATNO+SOFFSET).

NAME2 contains the ISDMAP entry number for an ISD and an index into the ISD to locate the second statement number entry. This word is present if IDENT2 is not NULL.

OFFSET2 contains the address of the Polish string formed for the offset of the second symbol. This word is present if IDENT2 contains code 9C (STATNO+SOFFSET).

- Hexadecimal Address:



IDENT1 identifies the type of entry. The possible code for a hexadecimal address entry, and its meaning is:

24 - ADDRES Identifies a hexadecimal address.

IDENT2 identifies the existence of a hexadecimal address range. The possible codes and their meanings are:

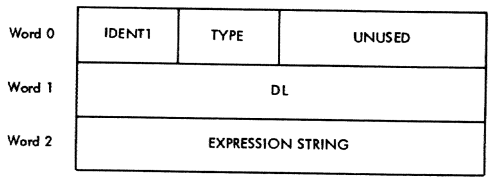
00 - NULL No range specified.

24 - ADDRES Identifies a hexadecimal address range.

DL contains the number of bytes to be displayed/dumped.

VMA contains the virtual storage address of the first hexadecimal address.

- Expression:



IDENT1 identifies the type of entry. The possible code for an expression entry and its meaning is:

28 - EXPRESS Identifies an expression.

TYPE identifies the data type of the expression result. The possible codes, and their meanings, are:

04 - ISDINT Identifies an integer expression.

05 - ISDREL Identifies a floating point expression.

06 - ISDCHC Identifies a character expression.

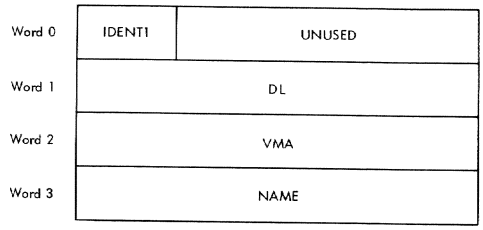
07 - ISDHEX Identifies a hexadecimal expression.

0E - ISDLOG Identifies a logical expression.

DL contains the data length of the expression result.

EXPRESSION STRING contains the address of the Polish string formed for the expression.

- Command Variable:



IDENT1 identifies the type of entry. The possible code for a command variable entry, and its meaning, is:

2C - COMVAR Identifies a command variable.

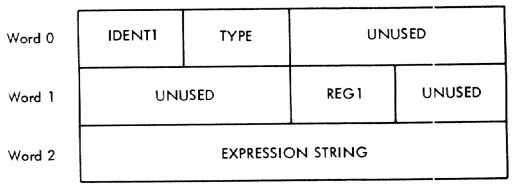
DL contains the data length of the command variable.

VMA contains the virtual storage address of the data in the combined dictionary entry.

NAME contains the virtual storage address of the combined dictionary entry.

SET CONTROL ROUTINE: An entry is made in the phrase list for each operand in the SET phrase. The format of an entry is determined by the syntax used to express the operand and the process used to evaluate it.

- Register:



IDENT1 identifies the type of register entry. The possible codes, and their meanings, are:

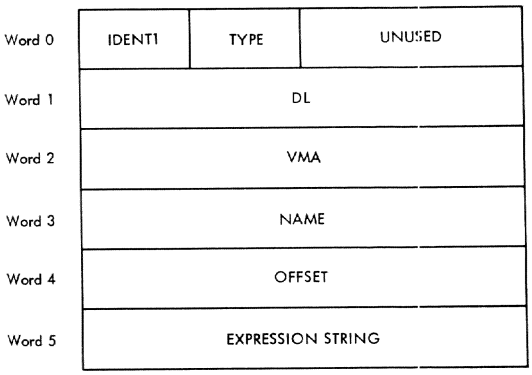
04 -	GENERAL	Identifies a general register.
08 -	SINGLE	Identifies a single precision register.
0C -	DOUBLE	Identifies a double precision register.

TYPE indicates the data type of the expression result.

REG1 contains the number of the register to be set.

EXPRESSION STRING contains the address of the Polish string formed for the expression.

- External Symbols:



IDENT1 identifies the type of entry. The possible codes for an external symbol entry, and their meanings, are:

14 -	EXTERN	Identifies an external symbol.
94 -	EXTERN+ SOFFSET	Identifies an external symbol with offset.

TYPE contains the data type of the expression result.

DL contains the number of bytes to be modified.

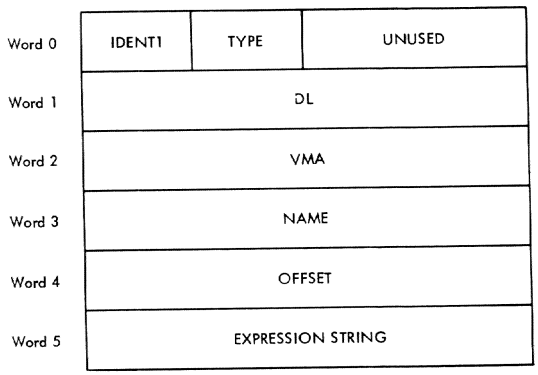
VMA contains the virtual storage address of the symbol.

NAME contains the address of the symbol definition in the PMD.

OFFSET contains the address of the Polish string formed for the offset. This word is present if IDENT1 contains code 94 (EXTERN+SOFFSET).

EXPRESSION STRING contains the address of the Polish string formed for the expression.

- Internal Symbols, Subscripted Arrays, and Statement Numbers:



IDENT1 identifies the type of entry. The possible codes for an internal symbol, subscripted array, or statement number entry, and their meanings, are:

18 -	INTERN	Identifies an internal symbol.
1C -	STATNO	Identifies a statement number.
20 -	ARRAY	Identifies a subscripted array.
98 -	INTERN+ SOFFSET	Identifies an internal symbol with offset.
9C -	STATNO+ SOFFSET	Identifies a statement number with offset.

TYPE indicates the data type of the expression result.

DL contains the number of bytes to be modified.

VMA is the virtual storage address of the symbol.

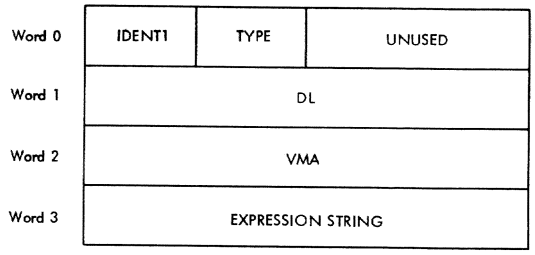
NAME contains the ISDMAP entry number for an ISD and an index into the ISD

to locate the defining symbol/statement number entry.

OFFSET is the address of the Polish string formed for the offset/subscript. This word is present if IDENT1 contains either code 98 (INTERN+SOFFSET), or 9C (STATNO+SOFFSET).

EXPRESSION STRING contains the address of the Polish string formed for the expression.

- Hexadecimal Address:



IDENT1 identifies the type of entry. The possible code for a hexadecimal address entry, and its meaning, is:

24 - ADDRES Identifies a hexadecimal address.

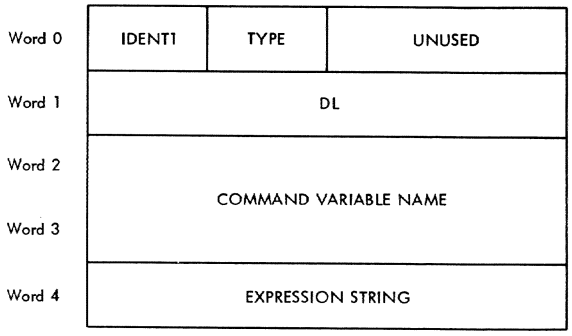
TYPE contains the data type of the expression result.

DL contains the number of bytes to be modified.

VMA contains the virtual storage address of the first byte to be modified.

EXPRESSION STRING contains the address of the Polish string formed for the expression.

- Undefined Command Variables:



IDENT1 identifies the type of entry. The possible code for an undefined command variable, and its meaning, is:

30 - UNDEFINE Identifies an undefined command variable.

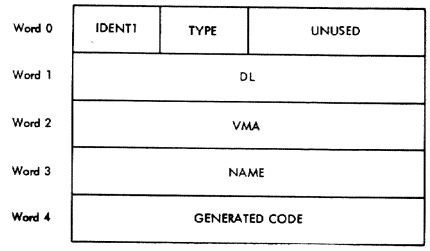
TYPE contains the data type of the expression result.

DL contains the data length of the expression result.

COMMAND VARIABLE NAME contains the 8 character name of the command variable being defined/redefined.

EXPRESSION STRING contains the address of the Polish string formed for the expression.

- PCS Input (Phase II) modifies an undefined command variable entry as shown below.



IDENT1 identifies the type of entry. The possible code, and its meaning, is:

2C - COMVAR Identifies a command variable.

TYPE indicates the data type of the expression result. It is used to define the entry code for the command variable.

DL contains the data length of the expression result. It is used to define the data length of the command variable.

VMA contains the virtual storage address of the data in the combined dictionary entry. For integer, floating point, and hexadecimal command variables, the data is initialized to zero. For character command variables, the data is initialized to blanks.

NAME contains the virtual storage address of the combined dictionary entry formed.

GENERATED CODE contains the address of a sequence of instructions that evaluate the expression.

CALL CONTROL ROUTINE: A single entry is made in the phrase list for the CALL phrase. This entry has the format shown below:

Word 0	VCONPTR
Word 1	NAMEPTR
Word 2	FLAGS
Word 3	NAME
Word 4	
Word 5	VCON
Word 6	RCON
Word 7	PARAMETER COUNT
Word 8	PARAMETER 1
Word 9	PARAMETER 2
Word 10	PARAMETER 3
Word 11	PARAMETER 4
Word 12	PARAMETER 5

VCONPTR contains the address of word 5, VCON.

NAMEPTR contains the address of word 3, NAME.

FLAGS contains various flags of importance to CZAMZ (USER CONTROL). When called by PCS this word is always zero.

NAME contains the name of the module being called.

VCON contains the virtual storage address of the module's entry point.

RCON contains the virtual storage address of the module's PSECT.

PARAMETER COUNT indicates the number of parameters contained in the list. It is initially set to zero, and is incremented by one for each parameter specified. The parameter count has a limit, imposed by PCS, of five.

PARAMETER x contains the address of the Polish string formed for the parameter.

BRANCH CONTROL ROUTINE: A single entry is made in the phrase list for a BRANCH phrase. The format of an entry is determined by the syntax used to express the operand, and the process used to evaluate it.

External Symbol:

Word 0	IDENT1	
Word 1	VMA	
Word 2	NAME	
Word 3	OFFSET	

IDENT1 identifies the type of entry. The possible codes for an external symbol, and their meanings, are:

14 - EXTERN	Identifies an external symbol.
94 - EXTERN+ SOFFSET	Identifies an external symbol with offset.

VMA contains the virtual storage address of the symbol.

NAME contains the address of the symbol definition in the PMD.

OFFSET contains the address of the Polish string formed for the offset. This word is present if IDENT1 contains code 94 (EXTERN+SOFFSET).

Internal Symbols and Statement Numbers:

Word 0	IDENT1	
Word 1	VMA	
Word 2	NAME	
Word 3	OFFSET	

IDENT1 identifies the type of entry. The possible codes for an internal symbol or statement number entry, and their meanings, are:

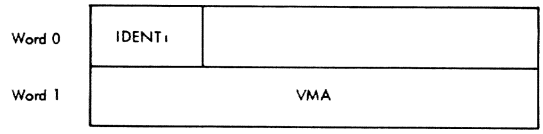
18 - INTERN	Identifies an internal symbol.
1C - STATNO	Identifies a statement number.
98 - INTERN+ SOFFSET	Identifies an internal symbol with offset.
9C - STATNO+ SOFFSET	Identifies a statement number with offset.

VMA contains the virtual storage address of the symbol.

NAME contains the ISDMAP entry number of an ISD, and an index into the ISD to locate the defining symbol/statement number entry.

OFFSET contains the address of the Polish string formed for the offset. This word is present if IDENT1 contains either code 98 (INTERN+SOFFSET) or 9C (STATNO+SOFFSET).

Hexadecimal Address:



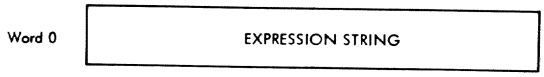
IDENT1 identifies the type of entry. The possible code for a hexadecimal address entry, and its meaning, is:

24 - ADDRES Identifies a hexadecimal address.

VMA contains the hexadecimal address.

GO CONTROL ROUTINE: The GO control routine produces no phrase list entries. The phrase list for a GO statement consists of a phrase list header only.

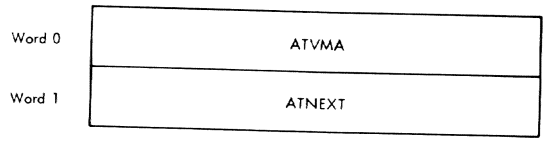
IF CONTROL ROUTINE: A single entry is made in the phrase list for an IF phrase. The format of the entry is shown below:



EXPRESSION STRING contains the address of the Polish string formed for the expression.

STOP CONTROL ROUTINE: The STOP control routine produces no phrase list entries. The phrase list for a STOP statement consists of a phrase list header only.

AT CONTROL ROUTINE: An entry is made in the phrase list for each operand in the AT phrase. The format of the entry is shown below:



ATVMA contains the virtual storage address of the operand. If the operand is expressed as a symbol with offset, code is generated to evaluate the offset. The code is then executed and the virtual storage address of the symbol is incremented by the offset. The resultant VMA is stored in ATVMA.

ATNEXT indicates the next STATAB entry to be processed. It is initialized to zero.

Phrase List Processing (Phase II)

PCS Input (Phase II) locates the phrase list(s) formed by Phase I. The identification of each phrase list and phrase list entry is inspected. If the list or entry contains a pointer to a Polish string, code is generated to evaluate the Polish string and the address of the generated code overlays the address of the Polish string.

If the statement is dynamic, the AT phrase list is processed after the generation of necessary code. A LOCTAB entry is then formed for each entry in the AT phrase list. An SVC is associated with the LOCTAB entry and is stored in the user's program. The LOCTAB entry is linked to the current STATAB entry and the AT phrase list entry is linked to the LOCTAB entry.

If a LOCTAB entry has already been formed for the AT phrase list entry of the current dynamic statement, it was formed as the result of a previous dynamic statement. In this case, the STATAB/AT phrase list chain is followed, to locate the last AT phrase list entry in the chain which is then linked to the current STATAB entry. The current AT phrase list entry is linked to the LOCTAB entry.

The linkage relationships between LOCTAB, STATAB, and AT phrase list entries, is shown in Figure 39 for the following example:

```
Statement 1:  AT  A
Statement 2:  AT  A,B,C
Statement 3:  AT  A,D
Statement 4:  AT  B,D
```

POLISH STRING (POLISH)

Polish String Organization

PCS Input (Phase I) forms a Polish string for each expression. Entries in a Polish string consist of: a header entry for the expression, an entry for each operand and operator in the expression, and a trailer entry to control the processing of the Polish string. If an operand is subscripted or offset, the operands and operators of the subscript/offset Polish string are included in the Polish string for the expression.

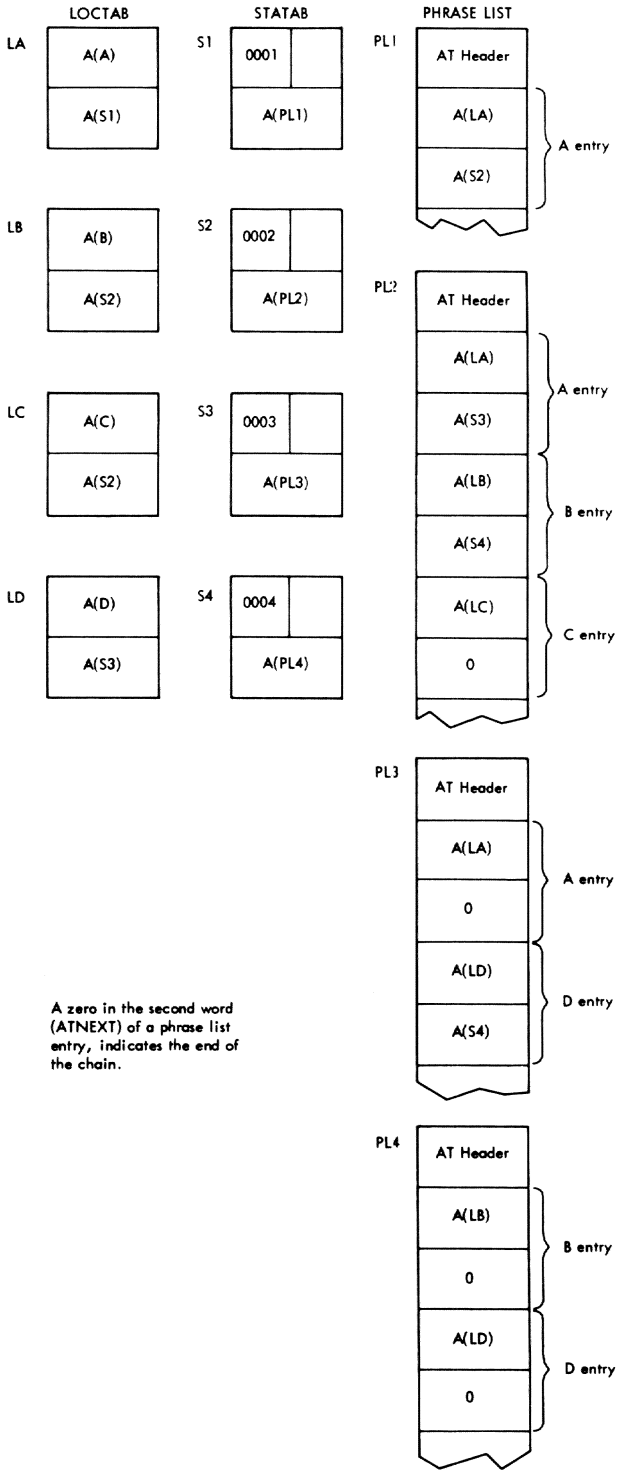


Figure 39. Linkage relationships between LOCTAB, STATAB, and Phrase Lists

The header entry contains information pertinent to the entire expression, such as, the type of arithmetic and registers to be used in the evaluation of the expression, and the data type and length of the expression result.

The trailer entry is used for control and indicates either the end of the Polish string, or a continuation of the Polish string into another page of storage.

An example of a Polish string formed for the expression B+C is shown in Figure 41. Note that this example assumes a Polish string extending only one page in length.

Since the operators, AND (&) and OR (|), separate logical expressions, it is necessary to have a header entry for each expression. For example, in the expression: A=B & C=D, one header would be formed for the expression A=B, and another for the expression C=D.

Since all expressions within subscripts and offsets must be evaluated in general registers, and the length of the subscript/offset result is always four bytes, it is not necessary to form a header entry for a subscript/offset Polish string.

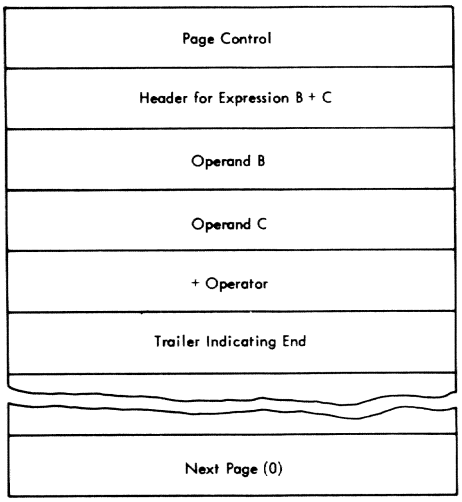
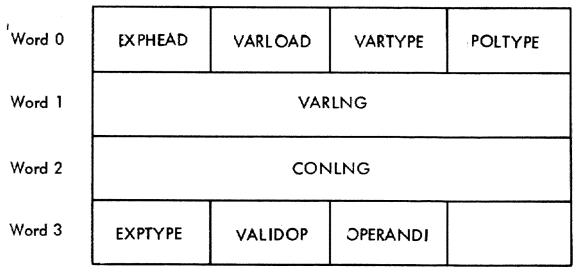


Figure 40. Sample Polish String

Note: Evaluation of subscripts and offsets is performed using logical arithmetic wherever possible. An Add Logical or Subtract Logical will be generated for plus and minus operators rather than an Add or Subtract. In the evaluation of a subscript expression (which is really an integer expression), a fixed point overflow cannot occur. This error is diagnosed as a dimension check error.

POLISH STRING HEADER: A Polish string header for an expression has the form:



- EXPHEAD identifies this entry as a Polish string Header. The possible code for this entry and its meaning, is:

  84 - POLHEAD Identifies a Polish String Header.

VARLOAD contains the variable load indicators which are used to determine whether or not the expression can be evaluated. The indicators are also used in selecting the appropriate prompt for type diagnostics. These indicators are initialized to zero and are selectively modified in the formation of the Polish string. The possible codes, and their meanings, are:

01 - NONFP Indicates that the expression is not loadable in a floating point register. This indication is set if a variable is not 4 or 8 bytes in length. It is also set if an integer constant, character constant, or hexadecimal constant is encountered in the expression.

02 - NONGP Indicates that the expression is not loadable in a general purpose register. This indication is set if a variable is not 1, 2, or 4 bytes in length. It is also set if a character constant or floating point constant is encountered.

04 - NONST Indicates that the expression cannot be evaluated in a storage to storage process. (i.e., the operands must be loaded into a register in order to evaluate the expression). This indication is set under the following conditions: If an arithmetic operator (*/+-) is encountered in the expression; if an integer constant or floating point constant is encountered; if all variables in the expression are not the same length; or if the length of the expression result is not equal to the length of the receiving operand (i.e., the operand on the left of the equal sign in a SET phrase).

08 - NONALIGN Indicates the presence of an unaligned variable. This indication is set if a variable is not aligned on the appropriate word boundry consistent with the length of the variable.

VARTYPE identifies the data type of defined variables. It is initialized to 00 (UNDEFINED). The data type of the first defined variable encountered is entered into VARTYPE. The data type of subsequent variables must agree with the contents of VARTYPE. If a variable type disagreement occurs, VARTYPE is set to FF (ERROR), a diagnostic is issued and the user is prompted for type definition.

POLTYPE identifies the data type of the expression. It is initialized to 00 (UNDEFINED). The data type of the first constant is stored in POLTYPE. All subsequent constants must agree with POLTYPE or the expression is rejected. VARTYPE, if defined, must agree with POLTYPE or a data type diagnostic is issued. (Note: POLTYPE specifies the data type of a command variable established through a SET phrase. POLTYPE also specifies the output format of an expression, if the result is to be displayed/dumped.)

VARLNG specifies the byte length of the expression result. It is initially set to zero. In the process of forming a Polish string, it is set to the byte length of the longest variable. If the expression does not contain a variable, VARLNG is set to the length of the receiving oper-

and. If the length of the receiving operand is undefined, VARLNG is set to the value contained in CONLNG. (Note: VARLNG contains the data lenght of a command variable established through a set phrase. VARLNG also indicates the length of a parameter in a CALL phrase and the length of an expression in a DISPLAY or a DUMP phrase.)

CONLNG specifies the byte length of the longest constant in the expression. Integer constants, address constants and single precision constants are always four bytes in length. Double precision constants are always eight bytes long. CONLNG is set to the byte length of the converted result of character and hexadecimal constants. Any padding characters involved in the internal processing of character or hexadecimal constants, are not included in the length indication contained in CONLNG.

EXPTYPE contains the indicators which identify the operators encountered. These indicators are initialized to 00 and are selectively modified during the formation of the Polish string. The possible codes, and their meanings, are:

01 - EXPARI	Identifies the occurrence of an arithmetic operator (+,-,*, or /).
02 - EXPREL	Identifies the occurrence of a relational operator (=,¬=,>,¬>, ≥,<,¬<, or ≤).
04 - EXPLOG	Identifies the occurrence of a logical operator (&,|,or).

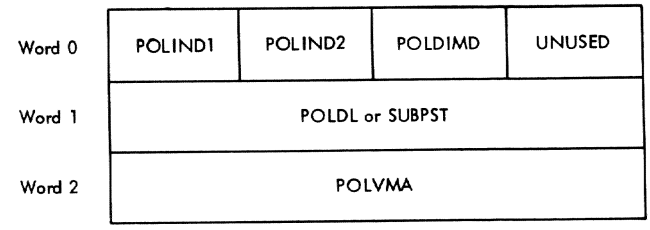
For the second header formed for an expression, these indicators are initialized to EXPLOG.

VALIDOP contains the valid operator indicators. These indicators have the same values and meanings as described in EXPTYPE (EXPARI, EXPREL, and EXPLOG). These indicators are initialized to one to indicate that all operators are valid. When an operator is encountered, the indicator for that particular operator type is checked to determine whether or not the current operator is valid. If the operator is deemed valid, the indicators are selectively modified as follows:

Operator Type	Action
Relational	If a relational operator is encountered, subsequent relational operators are considered invalid. VALIDOP is set to EXPLOG and EXPARI.
Arithmetic	If an arithmetic operator is encountered, a check is made to determine whether or not a relational operator is valid. If a relational operator is valid, logical operators are considered invalid and VALIDOP is set to EXPREL and EXPARI. If a relational operator is deemed invalid, VALIDOP is not modified.
Logical	If a logical operator is encountered, all operators are considered valid and VALIDOP is set to EXPLOG, EXPREL, and EXPARI.

OPERANDI indicates when an operand is present. It is set when an operand is encountered and cleared when an operator is encountered. If this indicator is not set when an expression or subexpression ends, a diagnostic is issued.

- Polish String Operand Entry: A Polish string operand entry has the following form:



POLIND1 contains miscellaneous Polish string indicators. The possible codes, and their meanings, are:

01 - POLLOG Identifies a logical unuary indicator. If POLLOG is set in an operand entry, the operand is to be logically negated before combining it with any other operand. If POLLOG is set for an operator entry, the result of combining the two operands by the operator is to be logically negated.

02 - POLARITH Identifies an arithmetic unary indicator. If POLARITH is set in an operand entry, the operand is to be arithmetically negated before combining it with another operand. If POLARITH is set for an operator entry, the result of the two operands combined by the operator is to be arithmetically negated.

04 - POLCON Identifies a constant operand.

08 - POLDIM Identifies a dimension operand of a subscript Polish string.

10 - POLOPT Identifies an operator entry. If this indicator is not set, the entry is considered to be an operand.

40 - POLSUB Identifies an operand entry of Polish string formed for a subscript/offset.

80 - POLCTRL Identifies a control entry. This indicator is set for a Polish string header, a continuation trailer, or a termination trailer.

POLIND2 contains miscellaneous Polish string indicators. The possible codes and their meanings, are:

01 - POLINREG Identifies an operand in a register. This indicator is set when code has been generated to load the operand into a register.

02 - POLSTORE Identifies a stored subexpression. If an operand must be loaded into a register, a data register is assigned. If, however, all data registers are in use, code is generated to store the contents of a data register in the PCS PSECT. The operand entry for the data register stored is identified as a stored subexpression.

04 - POLBASE Identifies an operand entry whose virtual storage address has been changed to a base and displacement.

40 - ISDDUM Identifies a dummy variable operand entry. This bit is obtained from the data type field of the defining internal symbol dictionary entry for the operand.

80 - POLALIGN Identifies an operand entry whose virtual storage address is unaligned. A dummy variable operand and an offset operand are assumed to be unaligned.

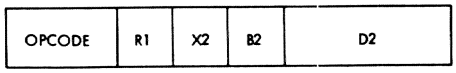
POLDIMD contains the dimension factor displacement (an index into the ISD entry for an array). An ISD entry consists of five words plus an additional word for each dimension in the array. For the first dimension entry formed, this index is initialized to 16 (i.e., an index from the first byte of the ISD entry to the fifth word, which represents the length of an array element). For subsequent dimension operand entries, this index is incremented by 4 to shift to the next dimension factor.

POLDL contains the data length of the operand.

SUBPST contains the base address, for a dimension operand, of the PSECT for the compiled module which produced the ISD. SUBPST is used in SUBGEN to locate address constants when a dummy variable array has adjustable dimensions.

POLVMA contains the data location of a variable. If the operand is a constant, the constant will have been stored in the address contained in POLVMA. The length of a constant entry is POLDL+8 bytes in length.

In the process of generating code, POLVMA is subject to modification. It has the following format:

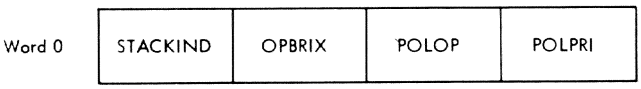


When a base register is assigned, (via GETBASE), for the virtual storage address of the variable, the page containing the address is assigned to a base register, the number of which is stored in B2, and the page increment is stored in D2. For dimension factors obtained from the ISD, the assigned base register contains the virtual storage address of the ISD entry for the array. The D2 field is set to POLDIMD of the operand entry. The POLBASE indicator is set to idnetify the fact that POLVMA is in base/displacement form. If a subscript or offset is to be applied, the number of the register containing the subscript/offset is stored in the X2 field (SUBGEN).

When a data register is assigned to the operand, GETREG stores the number of the register assigned in the R1 field. The command to load the operand is generated; all fields, excepting the R1 field, are cleared; and the POLINREG indicator is set.

When the contents of a register must be stored, GETREG generates the code necessary to store the number of the register in the R1 field in the PCS PSECT. The B2 field is set to identify the register covering the PCS PSECT (register 13), and the D2 field is set to the displacement into the PSECT. The indicator, POLSTORE, is set to identify a stored subexpression.

POLISH STRING OPERATOR ENTRY: A Polish string operator entry has the form:



STACKIND contains miscellaneous Polish string indicators. These indicators are discussed under POLIND1 of the operand entry discussion above.

OPBRIX identifies the operator. This operator index is eventually translated into actual machine code. The index values are:

0 - add or subtract operations.

1 - multiply operation.

2 - divide operation.

3 - compare operation.

4 - AND operation.

5 - OR operation.

POLOP contains the mask to be placed into branch-on-condition commands for relational operators.

POLPRI contains the operator priority indicator. This indicator is used in phase I. The operator priorities are:

0 - subscript/offset and base addition.

1 - multiply or divide.

2 - add or subtract.

3 - greater than, less than, greater than or equal to, less than or equal to, equal to, not greater than, not less than, not equal to.

4 - AND.

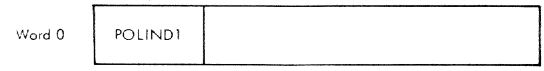
5 - OR.

6 - subscript comma.

7 - right parenthesis or terminator.

8 - left parenthesis.

POLISH STRING TRAILER: A Polish string header has the form:



POLIND1 identifies this entry as a Polish string trailer. The possible codes and their meanings, are:

81 - POLTERM Identifies a Polish string terminator.

82 - POLOVF Identifies a Polish string overflow (Continuation Trailer).

DISPLAY LIST (DISPLIST)

The display list (Figure 42), is created and used by the DISPLAY/DUMP subroutine. It resides in the DISPLAY/DUMP PSECT and is used for communication between subroutines.

The display list is divided into three main parts:

(1) The DISPLIST Header (words 0-4), which contains the information originally developed in housekeeping and NEXTLIST. This header is required by all DISPLAY/DUMP routines.

(2) The first item in DISPLIST, which contains the information pertinent to the data item (or first data item, in the case of a range).

(3) The second item in DISPLIST, which has the same format as (2) above, and is required to describe the last data item in a range. Information in (2) and (3) is gathered from the phrase list in NEXTLIST and from the ISD in NEXTISD.

DHFLAG identifies the first word of the header, and contains miscellaneous flags for the DISPLAY/DUMP subroutine. The possible codes, and their meanings, are:

04 - DOFFRNG Identifies an offset range. This indicator is set when an offset range is specified in the phrase list.

10 - DNOISDF Identifies the absence of an ISD. This indicator is set when an area between two ISD

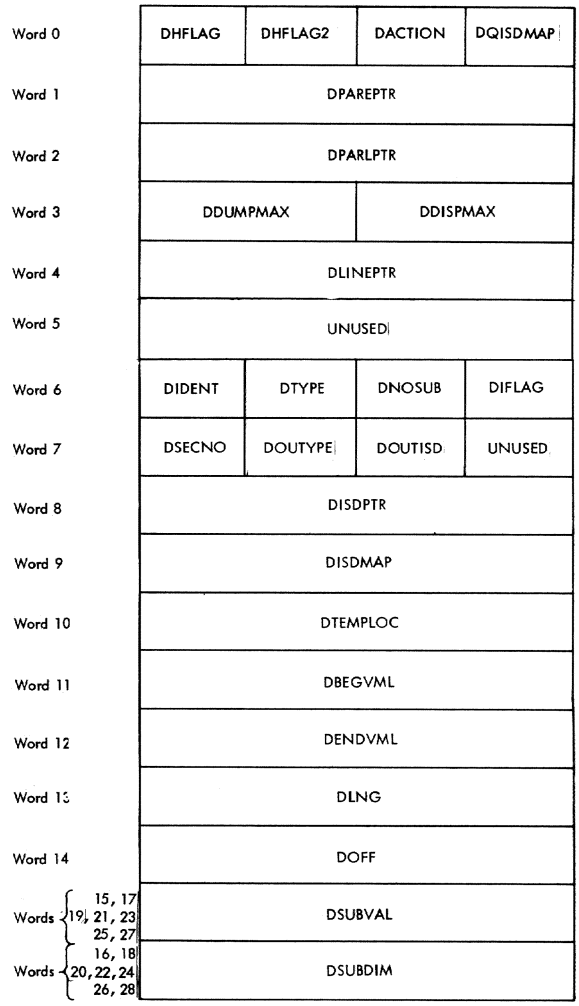


Figure 41. Display List (DISPLIST) Format

entries is being displayed.

40 - DINFORF Identifies an instruction format. This indicator is set to display a variable in the assembler format.

80 - DFTIMEF Identifies the processing of the first entry in a range. This indicator is on when processing the ISD for the first entry in a range, and off for second and subsequent entries in a range.

DHFLAG2 identifies the second word of the DISPLIST Header, and contains miscellaneous flags for the DISPLAY/DUMP subroutine. The possible codes and their meanings, are:

02 - DSVCF Identifies the presence of a PCSVC. This indicator is set when formatting an instruction which has been overlaid by a PCSVC.

04 - DDIAGNO Identifies the presence of a diagnostic. This indicator is set to signal diagnostics that occur in DUMP commands to be issued via GATE.

08 - DBYADJF Indicates that an adjustment is to be bypassed. This indicator is used in the formatting of hexadecimal ranges.

10 - DSTOPF Indicates that a diagnostic has been issued.

20 - DINHDRF Identifies the fact that an instruction header line has been DUMPed/DISPLAYed.

40 - DBR 2 Identifies a base register. This indicator is set when cross referencing a second symbol.

80 - DBR 1 Identifies a base register. This indicator is set when cross referencing the first symbol.

DACTION identifies the action to be taken by the subroutine. This indicator is set in housekeeping from information contained in the phrase list. It can indicate a Display, Dump or Set.

DQISDMAP qualifies the ISD map index. This indicator is set in housekeeping from information in the phrase list.

DPAREPTR identifies the phrase list entry. This phrase list entry pointer was originally passed as an argument from PCS Output, and points to the current position in the phrase list.

DPARLPTR identifies the phrase list. This phrase list pointer points to the end of the current phrase list.

DISPMAX indicates the maximum number of characters in a SYSOUT line. This indicator is initialized to 120.

DUMPMAX indicates the maximum number of characters in a PCSOUT line. This indicator is initialized to 120.

DLINEPTR indicates the next available position in a line.

DIDENT identifies the FROM entry. This indicator is set from information contained in the phrase list. See FLDID1 under Data Field Item.

DTYPE identifies the type of data to be dumped, displayed or set. This indicator is set from information in the ISD, or generated for non-ISD items. It is set from information contained in the phrase list, for a SET statement. See LOCTYPE under Data Location Item.

DNOSUB contains a subscript count which is set for arrays.

DIFLAG contains miscellaneous flags which have the following possible codes:

02 - DOFFLAG Identifies an offset item.

04 - DHEXADDF If the indicator is set, the location counter is formatted as 8 hexadecimal digits. If it is not set, the location counter is formatted as a 2 digit control section number and a 5 digit increment.

08 - DSUBFLG Identifies the subfield in an overlay as being processed.

20 - DQUALF Identifies a qualification flag. This indicator is set to suppress qualification of internal symbols.

40 - DLITEMF Identifies the last item. This last item indicator is set when the last item has been processed.

80 - DARRAYF Identifies an array.

DSECNO identifies the section number. This indicator is set from information contained in the ISD.

DOUTYPE is filled in from OUTYP in the Phrase List.

DOUTISD is filled in from ISDNO in the Phrase List. It is only used when a request for symbolic output is made in the form external symbol with offset.

DISDPTR is the ISD/FSD pointer. This indicator is set from the displacement in the phrase list and the ISD map.

DISDMAP contains the pointer to the ISD map. This pointer is set as a result of the search of the ISD map.

DTEMPLOC indicates the displacement of an item from the control section. This indicator is used for controlling the processing of the next ISD item.

DBEGVML indicates the beginning virtual storage location of the item. This indicator is set from information in the phrase list and adjusted by the offset or subscript, as required.

DENDVML indicates the ending virtual storage location of the item. This indicator is set from information in the phrase list and adjusted by the offset or subscript, as required.

DLNG indicates the length of the item. This indicator is set from information in the ISD, or generated for non-ISD items.

DOFF indicates the subscript/offset. This indicator is set as the result of executing generated code.

DSUBVAL contains the subscript value. This indicator is initialized to one for arrays. (See Note below.)

DSUBDIM contains the subscript dimension. This indicator is set for arrays, using the dimension factors contained in the ISD. (See Note below.)

Note: Space in the display list has been allocated for seven subscripts. This makes the total length of the item list 92 bytes and the complete list (consisting of one header and two display lists) 208 bytes in length.

Formats of displayed data correspond to the way the data field was specified in the PCS statement. Twelve specifications are defined, each with a unique IDENT value and entry format in the DISPLAY/DUMP phrase list. These are:

1. General register
2. Single precision register
3. Double precision register
4. % count
5. Internal symbol
6. FORTRAN statement number
7. Subscripted array
8. External symbol
9. Hexadecimal address
10. Expression
11. Command Variable
12. Offset (with 5, 6, 8, or 9 above)

SINGLE DATA LOCATIONS OR ARRAYS

1. Register -- will be displayed as follows:

   nR=FFFFFFFF

   General purpose register n in hexadecimal.

   nE=±.XXXXXXXXE±XX

   Single precision floating point register.

   nD=±.XXXXXXXXXXXXXXXXD±XX

   Double precision floating point register displayed in FORTRAN floating point format.

2. Internal symbol -- (Array or simple variable - same entry format) will be converted according to data type. These are:

   Integer (halfword)

   ±XXXXX

   Integer (fullword)

   ±XXXXXXXXXX

   Real (single precision)

   ±.XXXXXXXXE±XX

   Real (double precision)

   ±XXXXXXXXXXXXXXXXD±XX

   Complex

   (±.XXXXXXXXE±XX,±.XXXXXXXXE±XX)

   Logical

   TRUE or FALSE

   Address

   FFFFFFFF

   Immediate

   ±XXXXXXXXXX

   Instruction

   See item 7

   Character

   Actual character string

   Simple variables will be displayed as:

   A=value

   if a QUALIFY BY statement was used

   or

   PGM.A=value

   if not.

   Arrays will be displayed by rows. A new column will start at the

beginning of a line. For example, a 7 by 4 array is displayed as:

(1,1)

(5,1)

(1,2)

(5,2)

Equal lines will be suppressed with a comment:

(1,3):(7,4) CONTAINS (VALUE)

3. % count (may only be displayed as part of an AT statement) -- will be displayed as a halfword integer.

   Statement: AT [location];DISPLAY %

   Response (when location is reached): %=XXXXX

4. Hexadecimal -- data will be displayed in hexadecimal with an address given in hexadecimal.

   Statement: DISPLAY L'3CF'

   Response: 3CF = 9A

5. External or internal symbol with offset -- (same entry format) will be displayed in hexadecimal.

   Statement: DISPLAY A.(10)

   Response: A.(10) = 4F

6. Subscripted array -- will be displayed according to the data type as described.

   Statement: DISPLAY A(I,J)

   Response: A(X,Y) = value

   where X and Y are the current values of I and J.

7. FORTRAN statement number -- will be displayed as an instruction. Both FORTRAN statement instructions and assembler instructions will be displayed in the same format. The instructions will be converted to hexadecimal and symbolic, as in the assembler format. Operation codes will be replaced with the standard mnemonics, with assembler-extended mnemonics for branches. A symbolic reference will be made where possible, when instructions referencing storage have base registers defined in USING statements. No cross referencing will be attempted on FORTRAN instructions.

   Statement: DISPLAY NEXT

   Response: (headings are printed only once):

```
LOC      INSTRUCTION  LABEL  OPC  OPERANDS    SYMBOL
01 00022 4330 F042    NEXT   IC   3,66(0,15)  SWITCH
```

RANGES

Examples are given below where the from-to types are the same. All possible combinations are shown in Table 2.

1. Register -- general purpose registers and single precision floating point registers will be displayed in a series on a line. The order of display is the same as the order of input.

   Statement: DISPLAY 15: 0R

   Response: 15:0R FFFFFFFF FFFFFFFF

2. Internal symbols -- items will be converted according to the data type, as described. The form is:

   A:B

   where A and B may be either an array, a simple variable, or a statement number. The display format for A and B is as described for internal symbols. If the range includes areas for which a type is not known, those areas will be displayed in hexadecimal. Simple variables appearing between two instructions in a range will be displayed in the assembler DC format.

3. Hexadecimal -- if either data location is expressed with a hexadecimal number, the entire field will be displayed in hexadecimal. For example:

   DISPLAY L'103':L'127'

   will be displayed as (if the range required more than one line):

```
103      FF FFFFFFFF _______ _______
110 FFFFFFFF _______ _______ _______
120 _______ _______
```

aligned so that the second and following lines will start on a boundary that is a multiple of 16. Equal lines will be suppressed with a comment:

```
120:   1FF  CONTAINS 00000000
```

4. External or internal symbol with offset -- will be displayed in hexadecimal in the same format as described in 3, above. This also applies when this type is used with an internal symbol (either to or from).

5. Subscripted array -- will be displayed as shown for arrays, except that the limits will be the current values of the subscripts.

Internal symbols, statement numbers, and external symbols having an offset will be displayed in hexadecimal, in the same format as described in (3), above; this applies equally whether the offset is associated with the <u>from</u> or <u>to</u> location.

Table 2. Display Formats for Ranges

FROM \ TO	1	2	3	4	5	6	7	8	9	10	11	12	13*
1 general register	hex												hex
2 single precision		fp											fp
3 double precision			fp										fp
4 % count													int
5 internal					isd		isd					hex	isd
6 statement number						isd						hex	isd
7 array					isd		isd					hex	isd
8 external								hex				hex	hex
9 hex address									hex				hex
10 expression													exp
11 command variable													cd
12 (5,6 or 8) with offset					hex	hex	hex	hex				hex	hex

= range not permissable
* = format if no range is involved
cd = data type and length specified for entry in CSD
exp = format implied by expression type
fp = FORTRAN floating point format
hex = hexadecimal
int = halfword integer
isd = data type and length specified by user's ISD

APPENDIX H: PCS LIMITATIONS

The following restrictions were made, based on the amount of storage allocated for the various tables and working areas.

1. Maximum number of dynamic statements is 2^{16} -1.
2. PCS places no restriction on statement length.
3. Maximum offset is 2^{32}-1.
4. Maximum range is 2^{32}-1.
5. Maximum number of modules with ISDs is 255.
6. Maximum size of ISD is 2^{24} bytes.
7. Maximum character constant is 256 bytes.
8. Maximum hexadecimal constant is 512 hexadecimal characters.
9. Maximum number of operators in OPSTACK is 15.
10. Maximum length in set command is 256 bytes.

S/360 Program Control System Printed in U.S.A. GY28-2014-2

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
[U.S.A. only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]